AD-A115 499 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
CONTINUED DEVELOPMENT AND IMPLEMENTATION OF A STANDARD GRAPHICS--ETC(U)
DEC 81 P B TARBELL
UNCLASSIFIED AFIT/GE/EE/81D-58 NL

1 of 2
AD A115499

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/81D-58

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF A STANDARD GRAPHICS PACKAGE FOR THE AFIT VAX 11/780

THESIS

AFIT/GE/EE/81D-58 Philip B. Tarbell
CAPT USAF

AFIT/GE/EE/81D-58

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF A STANDARD GRAPHICS PACKAGE FOR THE AFIT VAX 11/780

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air Training Command
in Partial Fullfillment of the
Requirements for the Degree of
Master of Science

by

Philip B. Tarbell,III, B.S.E.E.

Capt USAF

Graduate Electrical Engineering

December 1981

Approved for public release; distribution unlimited

Preface

An AFIT thesis by Capt H. L. Curling was the first effort made towards the development of a graphics capability for the VAX 11/780 in the Digital Engineering Laboratory at AFIT. Capt Curling provided the design for a package of standardized graphics-input routines. This report describes the effort made to implement a package of standardized graphics-output routines on the VAX with which the input routines may be interfaced.

I would like to acknowledge the support of several individuals who were very helpful throughout this investigation. My thesis advisor, Dr. Gary Lamont, deserves thanks for providing the necessary guidance to keep me headed in the right direction. The members of my thesis committee, Professor Charles Richard and Capt Roie Black, were good sources of information and, along with Dr. Lamont, were helpful with their comments and criticisms of this report. Mr. Bill McQuay of the Avionics Lab deserves much appreciaton for all the support he has provided through the use of his VAX, without which this effort could not have succeeded. Most of all I want to thank my wife, Denise, who kept my spirits up during what has been a difficult period for both of us.

Contents

Contents

List of Figures

List of Tables

Abstract

Graphics packages that have been implemented based on the 1979 ACM/SIGGRAPH Core Standard Proposal were investigated. Two packages were identified that best met AFIT's needs, based on host processor implementation and availability. These two packages were GRAFCORE, from the Lawrence Livermore National Laboratory, and GWCORE, from The George Washington University. GRAFCORE was the package initially selected for installation on the AFIT VAX. GWCORE was also obtained, but not installed. Both packages currently provide output-only capability. Problems with GRAFCORE installation that had to be overcome included characters lost in transmission, incomplete files, and the need to manually convert some of the code from the LRLTRAN language used at Lawrence Livermore into standard FORTRAN. Future efforts in this area should begin with device driver design and implementation for both graphics packages, followed by implementation of graphics-input routines.

CONTINUED DEVELOPMENT AND IMPLEMENTATION
OF A STANDARD GRAPHICS PACKAGE
FOR THE AFIT VAX 11/780

I Introduction

The purpose of this thesis investigation is to continue development and implementation of a set of standard interactive graphics routines for the Digital Equipment Corporation VAX 11/780 located in the AFIT Digital Engineering Laboratory. These routines will interpret and execute graphics commands generated by function calls within a user program.

Interactive computer graphics is becoming an increasingly important area of computer operations. Real-time interaction between a user and a graphics program has many applications in education, engineering design, command and control, and management. The AFIT VAX was installed in 1980 and this computer should be provided with an interactive graphics capability so that it can be used, in part, as a graphics training and research facility. Since the use of interactive graphics is becoming more widespread in the Air Force, AFIT students need a resource from which to gain the experience in graphics techniques that they can then apply to current Air Force problems after graduation.

Historical Perspective

For a long time it has been accepted that computer graphics techniques can greatly improve the quality of man-computer interaction. Decreasing hardware costs have brought graphics display devices within reach of the average computer user, but software costs have not been as condusive to their widespread use.

Computer software is expensive to produce, and graphics software is no exception. One method of reducing software costs is to create standards for software. For example, a lot of effort has been devoted to standardization of FORTRAN, the creation of standard statistical packages such as SPSS, and to standard engineering programs [Ref 5:141]. As a result, non-graphics programs are routinely shared between computer installations, although not always without some difficulty.

The field of computer graphics has matured to where researchers are building on each other's work instead of investigating unknown areas. This has resulted in the evolution of some commonly accepted practices. These, in turn, lead to hardware that can be made more compatible, software that is more easily portable, communication protocols being established, and a better understanding among the people working in the field [Ref 11:365-366].

In 1974, a trend began to become evident of a growing interest in specifying a more precise definition of these common practices in order to get the maximum benefit from agreement. The formation of graphics standards would reduce duplication of effort in programming and provide for program portability and device-independence, as well as programmer portability.

Program portability means that computer installations can easily share graphics programs with minimal changes required in moving from one host system to another. Device-independence will allow the exchange of programs without the requirement that each site have identical hardware. This also means that, depending on the application, a user's program can be run on any of the graphics-output devices that are connected to the system without the need to rewrite the program for each device. In addition, new graphics equipment can be added to a computer system without having to change or rewrite a large number of applications programs; only a new device driver for the hardware will be needed. However, since a user will normally have a specific device in mind for his application program, how well it runs on other devices will depend on the purpose of the program and the capabilities of the various devices. In general, it is much easier for a program written for a storage tube display to run on a color raster-scan display than vice-versa.

Finally, graphics standards will allow easier training in the graphics field and make transition between computer sites easier for programming personnel. This is especially important in the Air Force, considering the high mobility of its personnel.

Background

Work on formulation of a graphics standard was first begun at a Workshop on Machine Independent Graphics, held at the National Bureau of Standards (NBS) in April of 1974 [Ref 20:1-2]. This workshop was sponsored by the Special Interest Group on Computer Graphics of the Association for Computing Machinery (ACM/SIGGRAPH). The result of the workshop was the establishment of the ground rules for a future standard. The scope of these rules was restricted to primitive functions for two and three-dimensional line drawings with text and viewing transformations for both passive and interactive graphics. Following the NBS meeting, a SIGGRAPH Graphics Standards Planning Committee (GSPC) was established to work on developing a standard, but no real gains were made towards that goal. The effort remained dormant until May of 1976, when a Workshop on Graphics Standards Methodology was held in Seillac, France and attended by representatives from Europe and North America. Out of this meeting came the concept of a two-part standards development: the development of a methodology and

the design of a graphics package.

The four main methodological themes that have been most influential are the following [Ref 20:2]:

1. Portability of programs (and programmers) is the most significant purpose of a standard.

2. Those issues which affect portability are those which affect application program structure, and therefore deserve the most attention.

3. Methodology of both design and use of a standard is as important as its semantics (functional capability). Syntax and specific calling sequences are much less important.

4. The functions of constructing and manipulating an object, and of producing a picture of the object, should be cleanly separated.

Working with these topics, the GSPC published their proposals in the fall of 1977 [Ref 19]. These proposals became known as the Core System.

As a result of comments, criticisms, changing interests, and new developments, a revised and extended version of the Core System was published in August of 1979 [Ref 20]. This proposal is being studied by groups such as the American National Standards Institue as a departure

point for refinement and ultimate adoption as a national and international standard. Although it is still only a proposal and not an adopted standard, the package is already being implemented, in one form or another, at several universities and government laboratories [Refs 2; 5:144].

An AFIT thesis by Capt H. L. Curling [Ref 3] provided the initial design work for a standard graphics package, based on the Core System, for the AFIT VAX. The Core System can basically be divided into two sections, input and output. The output section is necessary in both batch and interactive environments, but the input section is necessary only in the interactive environment. Capt Curling states that the batch environment is predominate in systems which are large enough to make standard graphics desirable, so attention has been channeled into design of output systems rather than input. He determined that the input section of the Core System had not been implemented or designed for any system, so he concentrated his efforts in the design of an input section for the VAX 11/780. His goal was to develop a design that could be integrated with an existing system of graphics output routines obtained from another source. This integration was to be the work of follow-on efforts.

Objective of this Investigation

The objective of this effort will be to integrate the graphics input design of Capt Curling with an existing

graphics output package and get the entire package operational on the AFIT VAX 11/780. A Tektronix 4014 storage-tube display and an ISC 8001G color raster-scan display are both available to be used as graphics-output devices. Device drivers will be required for both displays.

Capt Curling's input system was designed specifically with the Tektronix 4014 display in mind, as the ISC display was not available. Therefore, included in this effort will be any necessary expansion of his design to accommodate the capabilities of the raster-scan terminal.

Approach

This effort involves four major steps: the requirements definition, obtaining an existing graphics output package, implementation of device drivers for the output system, and integration of Capt Curling's input design with the output system.

The 1979 GSPC Status Report [Ref 20] will serve as the basic requirements document for this effort. The description of the Core System is quite extensive, and it is not expected that the entire standard will be implemented; that will be left for future efforts. The Core System can be implemented at any one of several different levels (see Chapter IV). The level of implementation for this project will depend on the characteristics of the graphics output

package that is obtained for the VAX.

A prime source for a graphics output package is the GRAFCORE/GRAFLIB package from the Lawrence Livermore National Laboratory (LLNL) in Livermore, California [Ref 8]. This package is currently running on a VAX at LLNL so it should be easily transferred to the AFIT VAX.

The initial implementation of the graphics output package will require device drivers for the Tektronix 4014 and the ISC 8001G. If Core System compatible drivers are not available for these devices, then modification of existing drivers will be explored before an attempt is made to design a driver from scratch. The built-in capabilities of each terminal will be considered in design of the drivers.

After the output system is made operational, the input system can be interfaced with it. Capt Curling's design for the 4014 may have to be expanded to take into account any additional capabilities of the ISC terminal. The two device drivers will then have to be expanded to accommodate the input system functions.

Finally, a basic applications program will be written to demonstrate the Core System routines on both of the graphics terminals.

Overview of the Thesis

This investigation involved an analysis of general requirements for interactive computer graphics and of the specific requirements put forth in the system requirements document (the Core Standard). Some existing standard graphics packages were evaluated and one was selected to be installed on the AFIT VAX. Listings of the routine names contained in this package, along with a description of their purposes, are contained in Appendices A and B.

The thesis is arranged by chapters which follow the general approach used in the thesis investigation. This chapter contains the introduction to the need for standardized graphics, supporting background and historical information, and the objectives and approaches involved in this investigation. Chapter II provides a discussion of basic standardized interactive graphics functions and Chapter III presents the requirements for a device-independent package based on these basic functions. An overview of the proposed Core Standard is presented in Chapter IV. Chapter V contains information on some of the standard graphics packages that are currently in use and Chapter VI details the efforts made to implement one of those existing systems on the VAX. Finally, Chapter VII presents the conclusions and recommendations of this investigation.

II Graphics System Principles

This chapter presents an overview of computer graphics principles in order to provide the necessary background information needed to establish the general requirements for a device-independent graphics package. These requirements will be presented in the next chapter. The discussion begins with a description of windows and viewports and how they are related to device-dependent and device-independent graphics packages. This is followed by brief descriptions of some of the basic function sets that can be included in a graphics package.

Device-Dependent Packages

The purpose of computer graphics is, of course, to generate visual images on a display device using input data from an application program. The most fundamental type of application program will produce images by directly addressing points in the display device's cordinate system. These points have integer values and can range, for example, from a 24x80 alphanumeric terminal to a 4096x4096 storage tube display, with 1024 addressable points in each axis being fairly typical. This method can be quite awkward when having to scale graphical data to fit into this fixed range or convert floating-point numbers into the integer format.

These problems can be easily avoided by defining all images in a coordinate system that is convenient to the user and then using a transformation to convert the user's coordinates into the specific device coordinates. The user's coordinate system is referred to as the "world" system. A window is defined in the world system that represents that part of the world system that is to be displayed on the output device. A corresponding viewport is defined on the output device that represents the area on the display surface in which the window contents will appear. This relationship is illustrated in Figure 1.

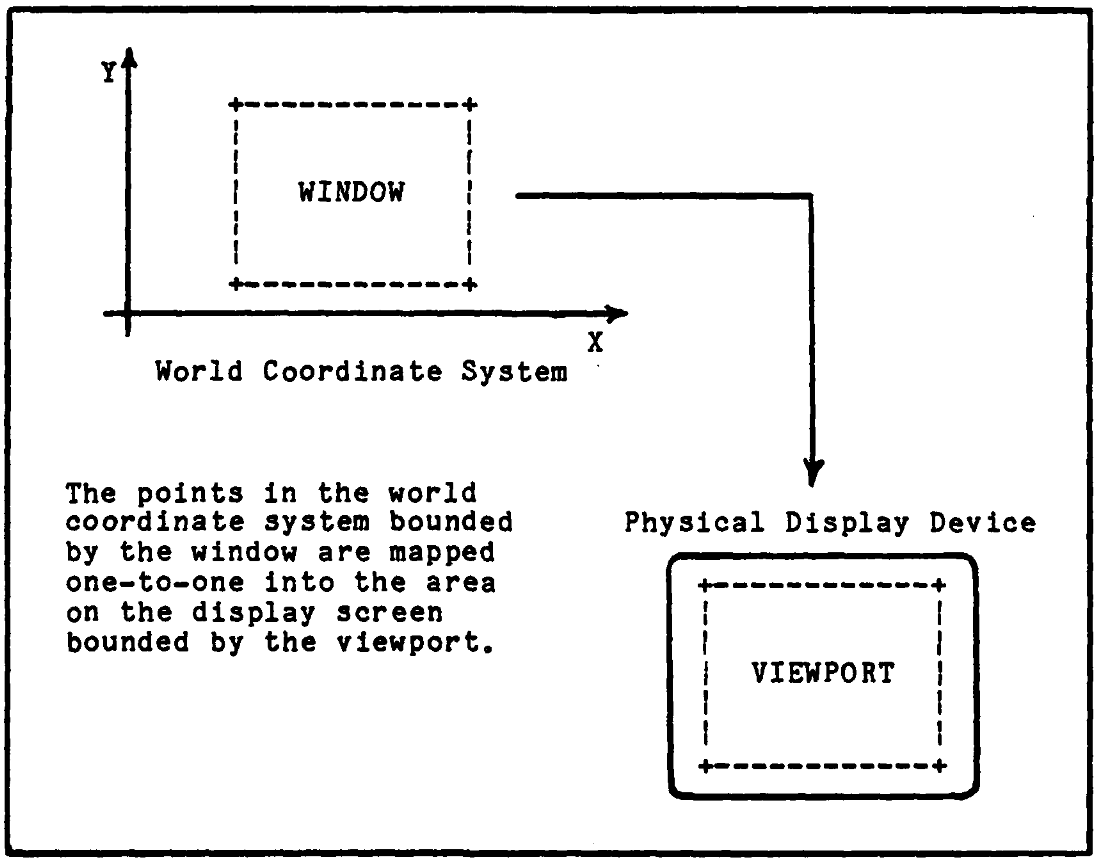


Figure 1. The Window and Viewport Relationship

Both the window and viewport sizes will affect the appearance of the displayed image. A very large window will result in a small image being displayed surrounded by a lot of space. A window only slightly larger than the defined image will produce an image that entirely fills the viewport. A window smaller than the image will result in display of only part of the image. These effects are shown in Figure 2.

The size, shape, and location of the viewport will have a corresponding effect on the size, shape, and location of the image, as shown in Figure 3. The relation between the image and the viewport is determined by the window and it stays constant as long as the window does not change; that is, an image that fills half the window *will also fill half* the viewport. Changing the viewport will not alter this relationship, so the image will always fill half the viewport whether the viewport is the same size as the display surface or one-tenth the size. However, if the viewport is not the same shape as the window (i.e., the window is square and the viewport rectangular) then the image will be distorted when mapped from the window to the viewport.

Even though these techniques isolate the programmer from the details of the specific graphics device, the package of graphics routines which perform this windowing transformation must still be highly device-dependent in

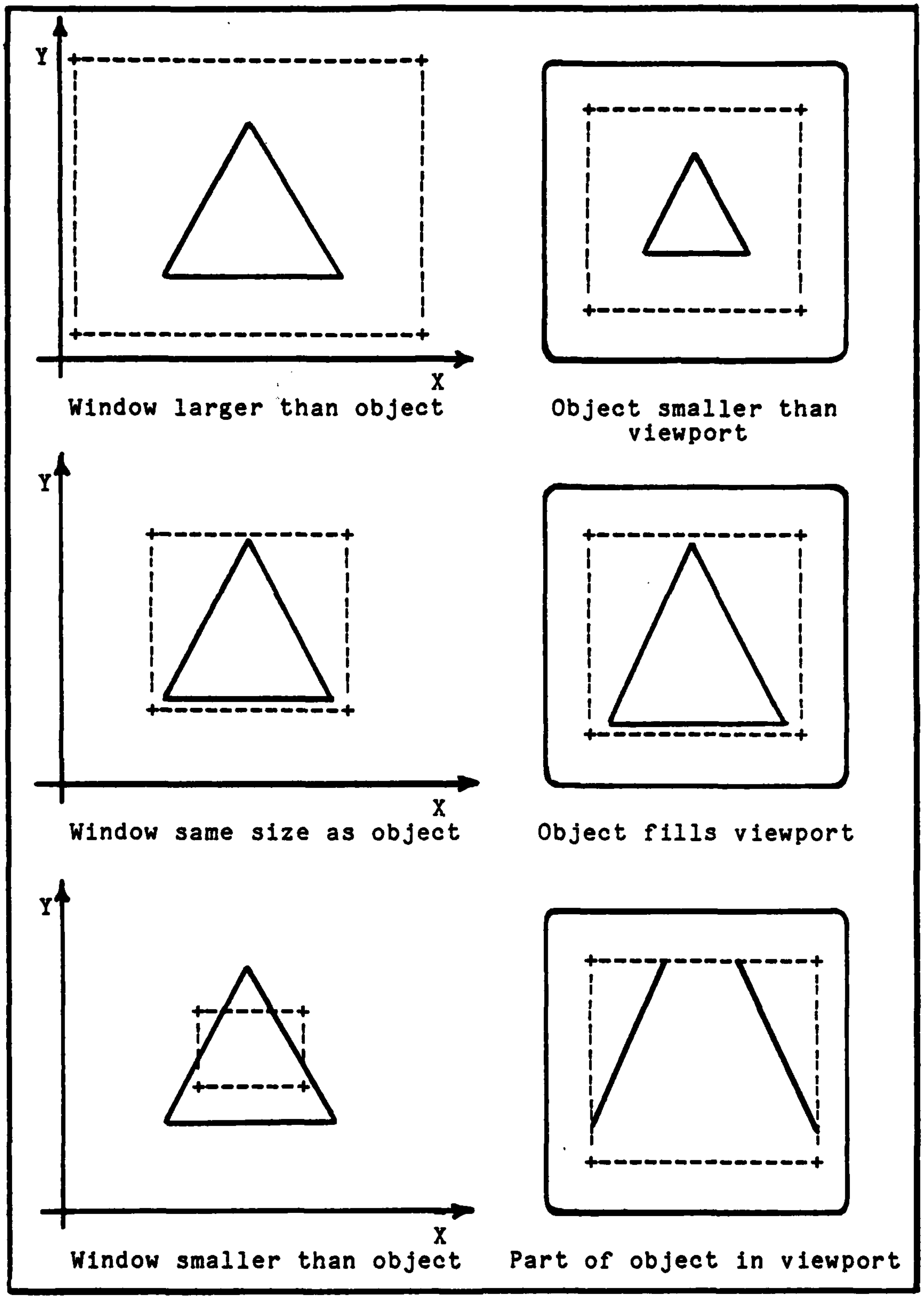
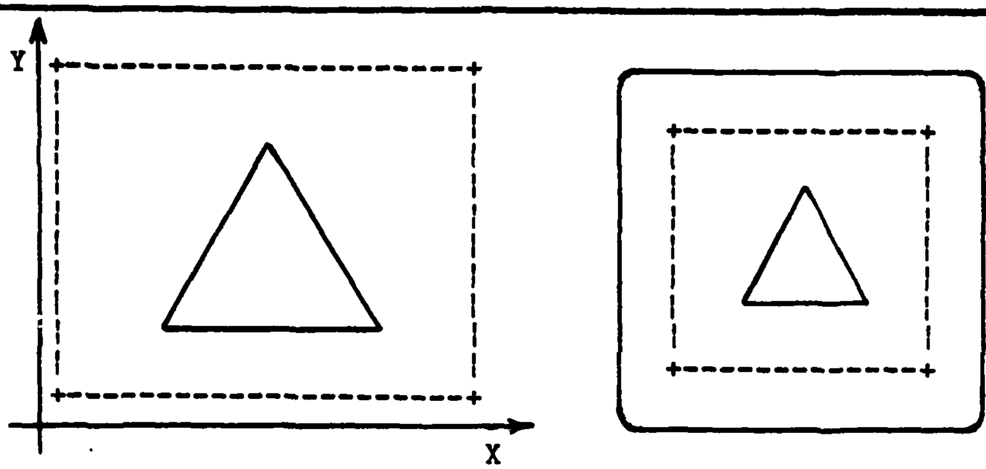


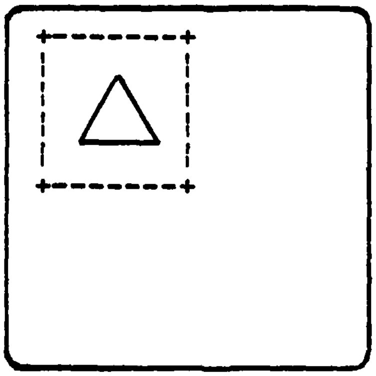
Figure 2. The Effect of the Window on the Displayed Image



Window in World Coordinate System

Viewport on Display Device

The contents of the window will be mapped one-to-one into the viewport. The appearance of objects in relation to the viewport will remain constant, whether the viewport nearly fills the display area, as shown in the top right figure, or only fills a portion of the display, as shown on the right.



When the viewport is not the same shape as the window, the resulting display will be distorted and will no longer be a true representation of the object as described in world coordinates.

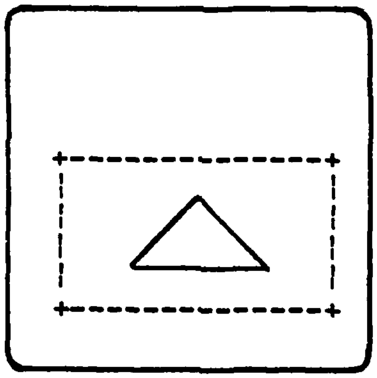


Figure 3. The Effect of the Viewport on the Displayed Image

order to convert world-coordinate commands into device-coordinate commands. A specific example of this is the Tektronix PLOT-10/Terminal Control System. The PLOT-10 System is a very powerful package designed mainly for drawing graphs and bar charts. Over 200 subroutines and functions are provided in this package to automatically generate many features required for drawing graphs, including the capability to graph in linear, log, or polar coordinates. While many of these routines are device-independent (at least in design, if not implementation), the basic image drawing routines must be device-dependent in order to drive the specific terminal devices such as the Tektronix 4010 and 4014.

Device-Independent Packages

To create a graphics package that is truly device-independent, an intermediate step must be included in the transformation from "world" to device coordinates. By converting from world coordinates to a device-independent coordinate system, followed by a conversion from the independent system to device coordinates, the device-independent operations can be effectively isolated from the device-dependent part.

A device-independent coordinate system that is easy to work with is the normalized device coordinate system. Figure 4 shows how the normalized system is used in a

graphics system. The normalized system has coordinates that range between 0.0 and 1.0 along each axis. This 1x1 coordinate system is a logical representation of the viewing surface of the actual physical device, no matter what that device may be. All output operations of the graphics package are performed with respect to this logical view surface: the viewports are all specified to lie within this 1x1 box and the coordinates of all drawing commands must be within the 0.0-1.0 range.

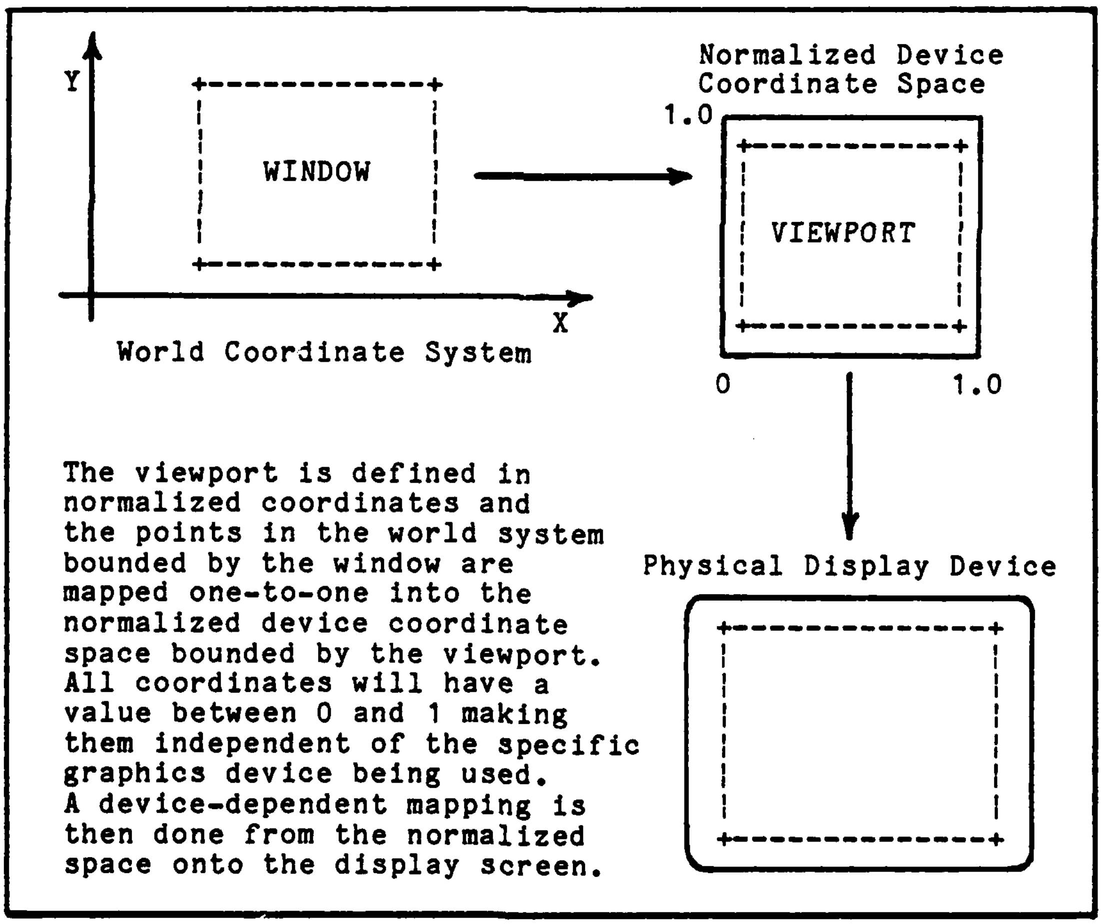


Figure 4. Use of Normalized Device Coordinates

The use of the normalized device coordinates has no great effect on the user or his application program. He can still create graphics images using world coordinates and has the same advantages provided by the windowing transformation as previously pointed out. In fact, the only way the user can notice any difference is in the specification of the viewport coordinates. These coordinates must now be specified in normalized coordinates instead of device coordinates. Other than that, the user need not be concerned with whether the graphics system is transforming everything into a 0.0-1.0 normalized coordinate system or into a 0-1023 device coordinate system.

The real advantage of using the normalized coordinates is that one program can be used to produce output on several devices without having to rewrite the program. Without a device-independent graphics package a user wanting to make the same drawing on several devices would have to write a separate program for each device. With a device-independent package, the output from a program is created in normalized device coordinates. This output can then be used to produce a display on any number of devices by simply using device drivers to transform the normalized coordinates into device coordinates. The drivers recreate on their display surfaces the drawing as specified on the logical view surface by the graphics package. Since the device drivers are independent of the application program, programs do not have to be

rewritten for each device. However, one factor that must be considered is the aspect ratio of the display surface. If the ratio is not 1-to-1, then the normalized space cannot be mapped onto the entire display surface, otherwise, distortion will result. The mapping must then be onto a smaller portion of the surface to maintain the proper ratio.

Graphics Function Sets

According to Newman and Sproull [Ref 12:81-82], the windowing functions defined earlier are one of only two function sets essential to all graphics systems. Windowing functions allow the programmer to choose his coordinate system and define the boundary of the visible portion of the picture without being concerned with the details of the device coordinate system. Graphic output primitives are the second essential function set. They are used to create the actual image by generating lines and text on the output device.

Other function sets that can be included in graphics systems are: segmenting functions, to permit easier modification of the picture; transformation functions for rotation, translation, and scaling; input functions that allow the user to interact with the program; utility functions that provide initialization and inquiry capability; and 3-D functions for producing three-dimensional views of objects.

__Graphics Output Primitives__. These form the basis of any graphics system, for without them, no images could be displayed on the screen. (As used here, screen refers to any output display device, whether plotter or CRT. Also, the terms beam and cursor will be used to represent the current drawing location even though the actual display device may use a pen or some other implement to produce images.) There are three general types of output primitives:

Move(X,Y) - sets the cursor to the location specified by coordinates (X,Y) without producing any visible output.

Draw(X,Y) - draws a line from the current cursor position to the location specified by coordinates (X,Y).

Text(S) - displays the string S starting at the current cursor position.

Other primitives are often included in a graphics system such as functions to draw points, circular arcs, or polygons. Additional functions can make the graphics package overly complex. Since most additional functions can be drawn using combinations of the above three primitives, extra functions are actually just a convience rather than a necessity.

These primitives will produce an output image, but they only tell the graphics system what to draw, not how it should look. There should also be a set of functions that allow the user to specify certain parameters of the displayed image. These can specify the intensity, color, linestyle (solid, dashed, or dotted lines), and certain text features such as style, size, and orientation.

<u>Windowing Functions</u>. These are an essential part of a graphics package. As described earlier, they isolate the user from the details of the specific display device being used. Only two functions are needed; one to define the window and the other to define the viewport:

Set_Window(WXMIN,WYMIN,WXMAX,WYMAX)

- define window position to extend from (WXMIN,WYMIN) at the lower left to (WXMAX,WYMAX) at the upper right.

Set_Viewport(VXMIN,VYMIN,VXMAX,VYMAX)

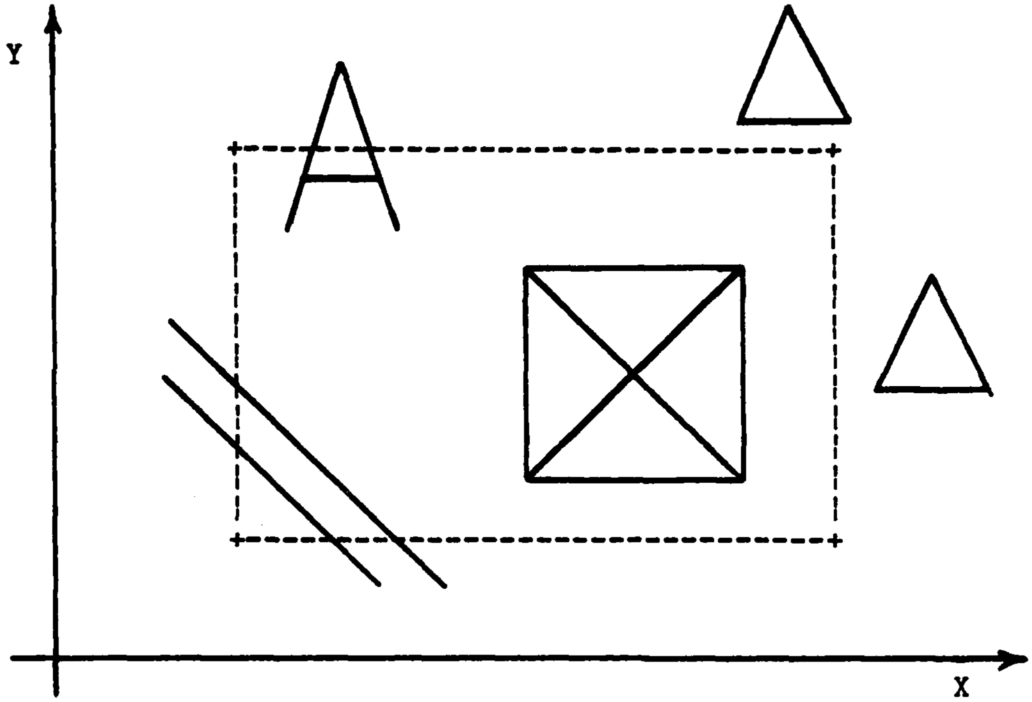
- define viewport correspondingly.

Associated with windowing functions is the clipping operation. Clipping prevents parts of an image that lie outside the window from appearing outside the viewport. Attempts to address coordinates that are outside the viewport can result in an overflow of the coordinate system of the display, which can result in undesirable effects such as wraparound. Even if the coordinates lie within the

screen boundaries, the unwanted parts could end up in an area of the screen that had been reserved for text. This can be just as annoying as unwanted text messages appearing in the middle of a drawing.

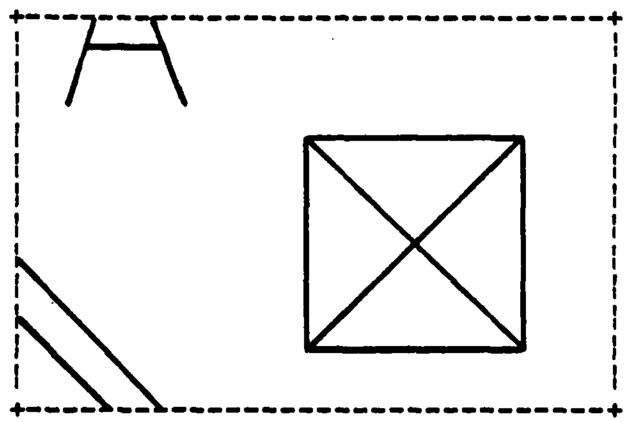
The first operation involved in clipping is to determine the visibility of a picture element with respect to the boundary of the clipping area. The picture element can be a point, line, curve, text character, or polygon. There will be three classes of elements: those that are completely visible and can be displayed; those that are completely invisible and can be ignored; and those that are partly visible and must be clipped. The basics of clipping are shown in Figure 5. Several algorithms have been developed for clipping lines and polygons [Ref 12:65-71] but they will not be described here.

Clipping is an internal operation of the graphics package and there are no parameters that the user would need to set with a special function. Normally, clipping is always enabled. However, there might be some situations for which it could be desirable to disable clipping. This could occur when the user knows that the entire picture will fit on the view surface without needing clipping or when all the clipping occurs in the application program. By being able to turn off clipping, much graphics system processing can be avoided. Therefore, some graphics packages may include a user function for turning clipping on and off.



Window in World System

- The box is completely within the window and will be displayed.
- The triangles are completely outside the window and will not be displayed.
- The 'A' and the diagonal lines cross the window boundary and will be clipped.



Viewport

Figure 5. Window Clipping

Segmenting Functions. Dynamic graphics frequently requires that only a small part of the picture changes while the rest remains unchanged. This is especially true of interactive graphics, where the operator may want to make small changes to a picture using a light pen or joystick. Erasing the screen to recompute and redraw the entire picture as every change is made is very inefficient. By dividing the displayed picture into subelements, changes do not have to be made to the entire picture; only one of these subelements, or segments, need be affected at any one time. The ability to make changes on a storage-tube display is not very efficient since the screen must be erased every time a new version of the picture is desired. But segmentation does help because those segments that are not affected by the change need only be redrawn; there is no need to recompute them as long as they have been saved in a display list. Raster-scan displays are capable of selective erasing of portions of the displayed image and, therefore, are more suitable for interactive graphics. Segmentation is what allows only parts of the picture to be selected for change.

To modify portions of the picture, the picture elements should have a way of being identified, or named, by the user. Each output primitive could be given a name, but in almost all cases the unit of modification consists of a group of several of these primitives and not just one point or one line. Since the variability of displayed pictures

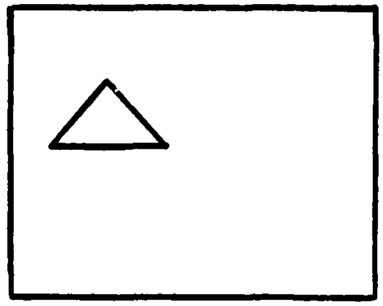
and user requirements is so great, it is impractical to specify a fixed-size modification unit. Instead, the user is allowed to specify in his application program the size of the units. In other words, the user determines into how many subelements his picture will be divided. This is done with the segmenting functions, of which the following are a basic set:

Open_Segment(N)	-	any currently open segment is closed and a segment named N is opened. All graphics primitives that are invoked while this segment is open are put into this segment.
Close_Segment	-	the currently open segment is closed.
Delete_Segment(N)	-	segment N is permanently removed.
Post_Segment(N)	-	segment N is made visible.
Unpost_Segment(N)	-	segment N is made invisible. The segment is still retained, so it can be made visible at any time until it is explicitly deleted.

Figure 6 gives an example showing the use of segments.

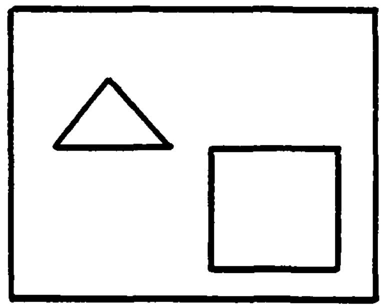
1. Display Triangle

Open_Segment (t)
.
(output primitive
description of triangle)
.
Close_Segment
Post_Segment (t)



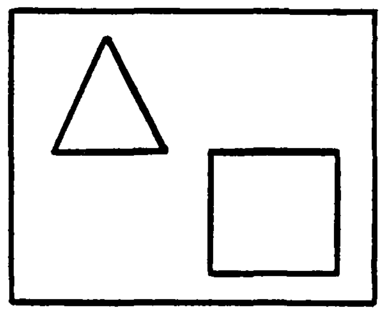
2. Display Square

Open_Segment (s)
.
(output primitive
description of square)
.
Close_Segment
Post_Segment (s)



3. Redefine Triangle

Open_Segment (t)
.
(output primitive
description of new
triangle)
.
Close_Segment
Post_Segment (t)



4. Delete Triangle

Delete_Segment (t)

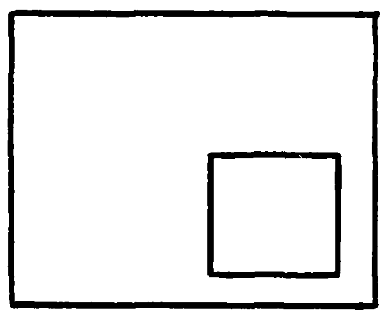


Figure 6. An Example of the Use of Segmentation [Ref 12:94]

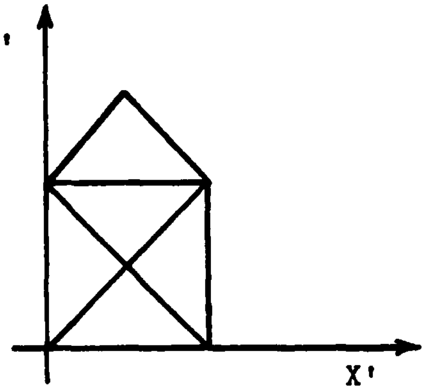
Transformation Functions. The mapping of coordinates from the world coordinate system to the normalized device coordinate system is a type of transformation that is referred to as a viewing operation. The viewing operation is just a linear mapping between two coordinate systems and is governed by the size of the window and viewport as previously described. There are two other types of transformations and they both make use of scaling, rotation, and translation. The first type is called an instance, or modeling, transformation and it is applied to graphics primitives as they are being invoked. The second type is an image transformation and it is applied to segments.

As has already been stated, the user describes images using some convenient coordinate system called the world system. Although the user describes images using world units, they do not necessarily have to be referenced to the origin of the world system; the user's own independent origin can be used. The advantage in doing this is that a user can create many views of an object but only have to describe it one time. The object is described in world units using graphics primitives as would normally be done except that the coordinates used are with respect to some convenient origin local to the object. This object description then becomes the "model" for that particular object. By using scaling, rotation, and translation functions this model can be recreated at any location, with

any size or orientation, to produce new views of the object in the world system. This is the instance transformation.

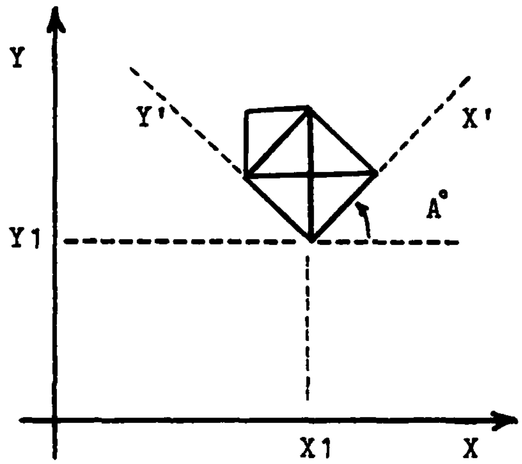
For example, suppose a diagram is being drawn that requires a special symbol to be drawn in several locations on the screen and that the symbols at each of the locations must be of a different size and orientation. Several graphics primitives will be needed to describe the symbol. Without instance transformations, these several primitives will have to be repeated for each appearance of the symbol in the diagram; each set of primitives will have different coordinates, depending on the exact location of the particular symbol. The use of instance transformations allows the user to describe the symbol only one time, creating what could be called a "template" for the symbol. The coordinates of the graphics primitives are in world units but with respect to some local origin, possibly the center of the symbol or the lower left corner. Then for each of the appearances of the symbol in the diagram, transformation functions can be used to "position" the template for the symbol at the proper location in the diagram and draw it with the proper size and orientation. Figure 7 illustrates this technique.

Image transformations are basically the same thing except that they are used on objects that have already been put into segments and displayed on the screen. They are used quite often for interactive graphics to allow the user



The user describes an object in world units with respect to an origin that is local to the object. This description can be put into a subroutine to form a standard "template" for the object.

For every appearence of the object in the world system, the subroutine description can be called and positioned using the scale, rotation, and translation functions. The example to the right illustrates the positioning of the object at location (X1,Y1) and rotated by 'A' degrees. The figure below shows how several versions of the object could appear in the world system.



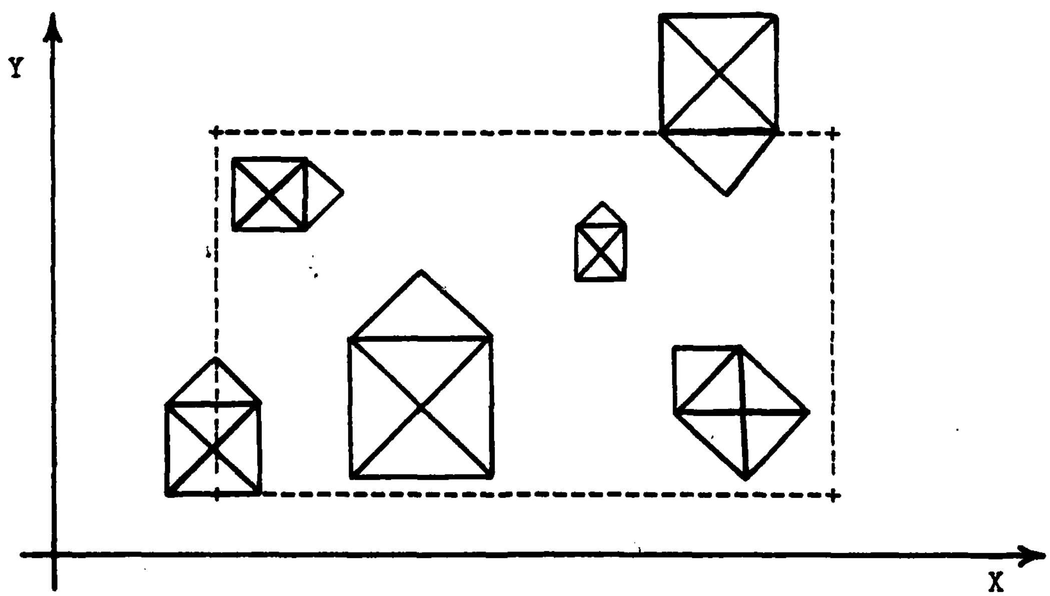


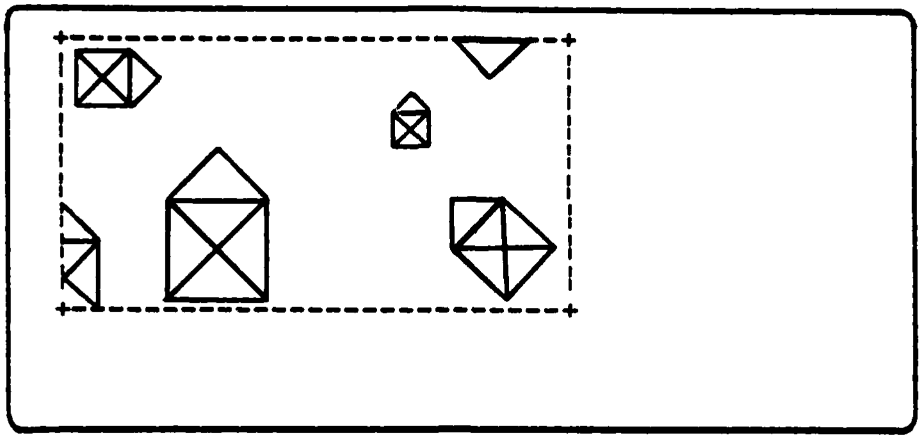
Figure 7. Instance Transformations

to change the appearance of drawings. Referring to the earlier example of a drawing with several similar symbols, once the drawing has been displayed, the user may not like what is displayed. If the program provides an interactive capability, a light pen, joystick, or some other input device can be used to rearrange the symbols into the desired positions. (This assumes that each symbol was put into a separate segment.) Since this transformation operates on objects that have already been displayed, clipping will no longer have any effect. This means that the objects can be moved to outside the viewport. It also means that moving a clipped image away from the viewport edge will not cause the clipped portion to suddenly reappear. These principles are illustrated in Figure 8.

The actual function names for the two types of transformations would have to be different, but each transformation would consist of the following three general functions:

Scale(SX,SY) - scale the object by factors SX and SY in the X and Y directions relative to the origin.

Rotate(A) - rotate the object through 'A' degrees about the origin. Whether the rotation is clockwise



Viewport on Physical Display Device

The figure above shows how the contents of the window in the world system from Figure 7 could appear on a display device.
By using the scale, rotation, and translation image transformations, the user can change the appearance of the display without having to redefine each object.
To individually change each object, each one must be defined within a separate segment.
The figure below shows one possible new configuration for the displayed picture. Notice that objects can be moved to outside the viewport and that clipped objects will remain clipped.

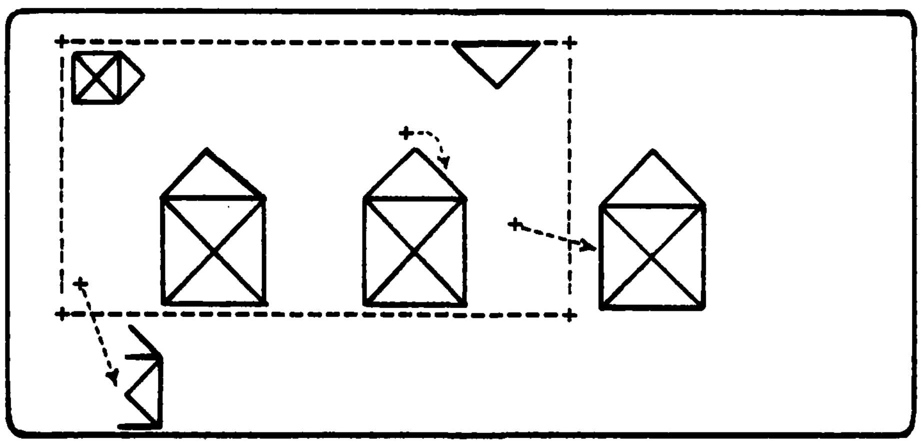


Figure 8. Image Transformations

or counterclockwise will depend on the graphics package implementation.

Translate(TX,TY) - translate the object through distances TX and TY measured in the X and Y directions.

A fourth function could also be provided that would allow specifying all three of the above functions at one time, such as Transform(SX,SY,A,TX,TY).

Input Functions. Input functions are necessary to allow interactive operation of graphics programs. Often, it is not enough to simply display a picture on a screen; there are many activities, such as computer-aided design, in which a user needs to be able to make changes to the displayed image in real time. These activities were discussed in the previous two sections on segmentation and transformations.

There are only two basic types of graphical interaction: pointing at items already on the screen, and positioning items - either locating new ones or repositioning old ones. Most display terminals provide an alphanumeric keyboard for input to the computer, but this is generally not adequate or very convenient for these applications. However, when a specialized graphical input

device is not available, input functions can be simulated in software to make use of the keyboard as the input device. Many special devices, such as the joystick, tracker ball, light pen, and tablet, have been developed to provide the necessary input capability. Some of these input devices are better at positioning than pointing, and others are just the opposite. Software can be used to compensate for any deficiencies in capabilities.

One of the most important features of graphics input is the need for visual feedback. Except for a light pen, which is pointed directly at the screen, input devices need an indication on the screen of the current location to which they are "pointing". This indication is normally provided by a small cross, or graphics cursor, that is displayed on the screen when the input device is active. The cursor can be made to move about the screen by operating the input device, thus allowing the operator to "point" to any spot on the screen. The more intelligent terminals will have this graphics cursor feedback built in; others will require computation and display of the cursor by the graphics software.

Another use of feedback is to indicate to the operator what part of the image has been selected. Once the cursor is moved to the proper location, the user will normally push a button either on the input device or on the keyboard to indicate the cursor is at the location for a specific action

to take place. This action could be to display a certain symbol or image at that point, to delete the image that is already displayed, or to select the image for moving. In the case of images that are already on the screen, the operator needs to know that the image he has in mind to select is the one that the graphics system actually does select. This feedback can be accomplished by blinking or highlighting the screen image, or even by drawing a box around it. In this manner, the operator can be sure of knowing exactly what part of the picture will be operated on before actually performing the operation.

There are many input techniques [Ref 12:159-181] that make effective use of input devices. Some of these techniques provide positioning aids to the user such as modular and directional constraints, gravity-field effects, scales and guidelines, rubber-band lines, and dragging. There are also techniques that aid in selection, whether it is the type feedback mentioned above or the display of a selection menu. A third category is inking and painting effects. These allow the operator to use the display as a drawing pad to "paint" images with varying size "brushes" and in different colors. Used in conjunction with some of the positioning constraints, these techniques can be used to produce high quality block diagrams and technical illustrations. The last input technique discussed by Newman and Sproull is the use of freehand input to draw characters

or symbols on the screen and then letting the computer recognize the characters and replace then with a neatly drawn symbol.

These input capabilities are not what you would want to include in a basic graphics package. They would add complexity to the package and would in some instances be too device-dependent to be easily implemented. Therefore, these functions are normally implemented as a higher level input package built upon a basic set of graphics functions.

As a minimum, input functions would be needed to detect when an input device was used, determine which device was used, determine some input value (for example, the coordinates of the graphics cursor), and provide some sort of control of when input devices can be enabled. Once the graphics system identifies the device that is used and gets a corresponding value to go with it, it is then up to the higher level input package to determine what to do with this information. Only the user program can determine that an input from a joystick while the cursor is at location (Xn,Yn) means that the operator is selecting menu item Mn. Another program that receives the same input conditions might want to have text inserted at location (Xn,Yn). A third program might want the symbol centered at (Xn,Yn) to be deleted. As these examples illustrate, one set of input parameters can indicate different actions to different programs, so all the basic input functions need to do is to

identify the source of the input and provide the coordinate data to go with it.

A main program may not always be sitting idle just waiting for input from the operator. As happens quite often, the program can be in the middle of a long calculation when the input data is received. To prevent having to interrupt the program, or keep from losing data, the input data can be entered into an event queue until the program is ready for input. The queue is a list of blocks, each representing one user action, or event. Each block contains the type of device causing the event and the contents of the device registers. The events are entered into the queue in the order that they are received and the program executes them in the same order. A function that can provide the necessary input event data is:

Get_Event(E) - return in record E the input data from the event on the top of the queue. If the event queue is empty, wait for an event to occur.

If input actions occur too quickly, the main program may not be able to remove events from the queue as fast as they are being entered and the queue can overflow. Even if it does not overflow, it will get farther and farther behind in processing the inputs. This can become evident on the

screen during such actions as dragging or rubber-band lines. To prevent this, a function can be used to specify when a certain type of input is to be allowed:

Permit_Event(T) - allow an event of type T to be added to the event queue.

In most cases, this function is used to allow only one event of any particular type to be in the queue at any one time.

Utility Functions. These functions play a supporting role in a graphics package. They do not directly affect the creation of an image for display, but they do assist the programmer in preparing the graphics system and the output devices.

Clear_Screen - clears the display device

Initialize - This is just a category of utitity functions, as there may be more than one use for this type of function. Probably the most common initialization function is the one used to initialize the graphics package at the start of program execution. This can be used to allocate buffer or file space,

initialize variables, set up default values for parameters, and initialize input or output devices. If there are several I/O devices available on a system, a separate initialization function may be provided for each device to provide more flexability to the programmer.

Inquire(parameters) - This is also a category of functions, as there probably will be several types of inquiry functions. These functions can be used to return to the program the values of the requested parameters. The requested data can be information about input or output device characteristics, the current cursor location, window or viewport dimensions, the current color or intensity being used, or any other system parameter that might be available.

Three-Dimensional Functions. Up until now, the functions discussed have been for two-dimensional graphics. Some of them, such as the segmentation functions, are

dimension-independent and will be the same for 3-D as they are for 2-D. Others, such as the Move and Draw functions, will only require the addition of a Z-coordinate to their parameter lists. There are also additional special functions required just for 3-D due to the extra complexity of three-dimensional graphics.

The same window-to-viewport functions are used to map 3-D images from world to normalized device coordinates as are used for 2-D images. However, before this mapping can take place, the 3-D images must first be converted into a 2-D representation that is suitable for output on a normal display device. This is accomplished by projecting the three-dimensional image onto a view plane. The resulting two-dimensional image can then be mapped into the normalized device coordinate space.

Two types of projection are used to convert the 3-D images into a 2-D view plane representation: parallel and perspective. In parallel projections (Figure 9a), each 3-D point is mapped onto the view plane by following a line that is parallel to a given direction of projection. In perspective projections (Figure 9b), the lines used to map the points onto the plane are not parallel but emanate from a specified center of projection. A line is drawn from this projection point through the 3-D point, and where it intersects the view plane is where the mapped point will be located.

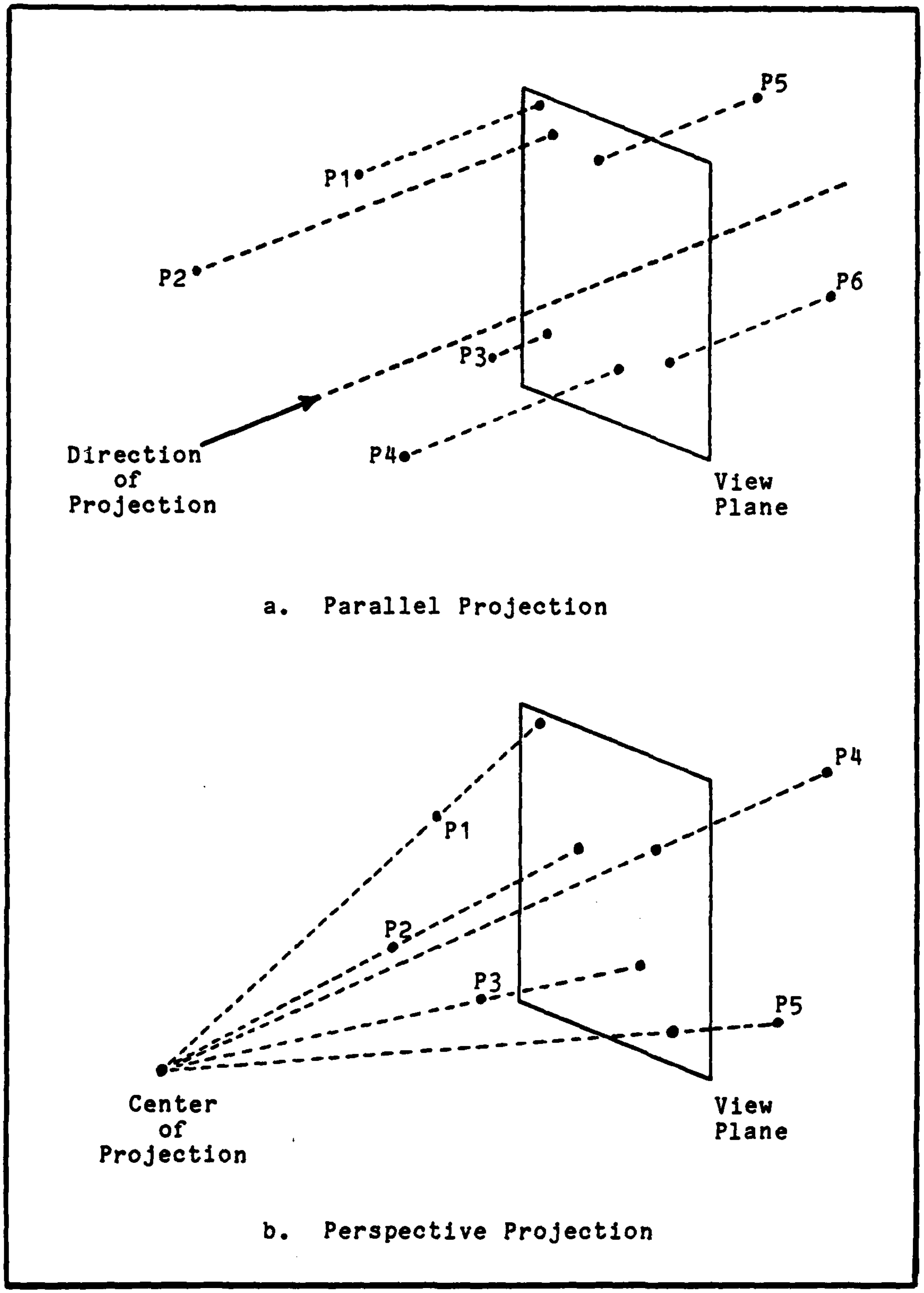
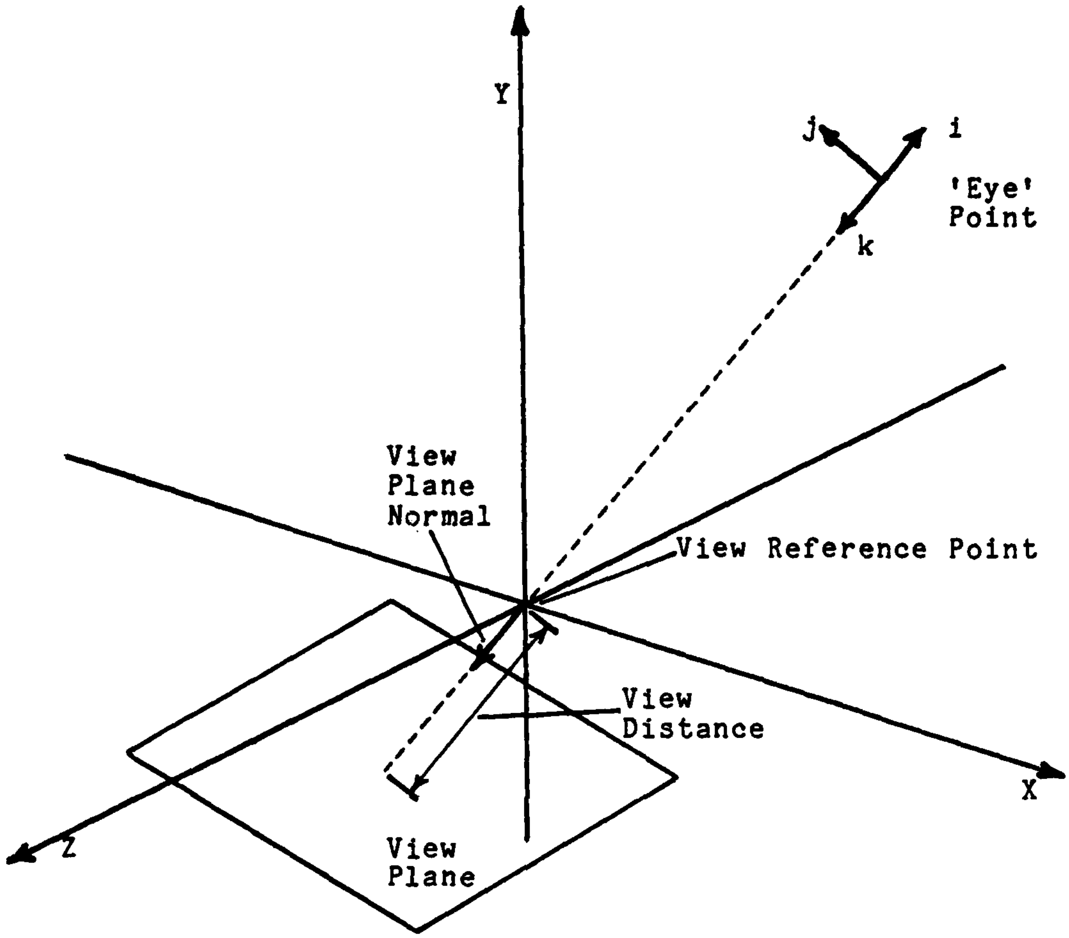


Figure 9. 3-D Projections

Two-dimensional images exist only in the X-Y plane and they can be viewed only from a point directly in front of the plane, just like you view the images once they are displayed on an output device. Three-dimensional images, however, can be viewed from any point in space. Therefore, the point in space from which the image is to be viewed must be specified. The first step is to select a reference point. This is typically a convenient (known) point in world coordinates that is on or near the object being viewed. The origin of the world system is often used as the reference point. Next, the view plane must be specified. Two functions are required to specify the view plane: the direction is given by specifying a normal to the plane, and its location is specified by its distance from the reference point. This is illustrated in Figure 10. Finally, the "up" direction on the view plane, the type of projection desired, and the location of the projection point must be chosen.

Set_View_Reference_Point(X,Y,Z) - the location of the reference point is defined in world coordinates by the values of X,Y, and Z.

Set_View_Plane_Normal(DX,DY,DZ) - DX,DY, and DZ define a unit vector in world coordinates relative to the view reference point. The view plane is perpendicular to this normal vector.



In this illustration, the origin of the world system is being used as the View Reference Point. The View Plane Normal is specified with respect to the View Reference Point and the View Plane is perpendicular to the View Plane Normal. The View Distance sets the distance of the View Plane from the View Reference Point in the direction of the View Plane Normal. The "Eye" Point is the projection point and is the origin of the i-j-k "Eye" Coordinate System.

Figure 10. The 3-D View Plane

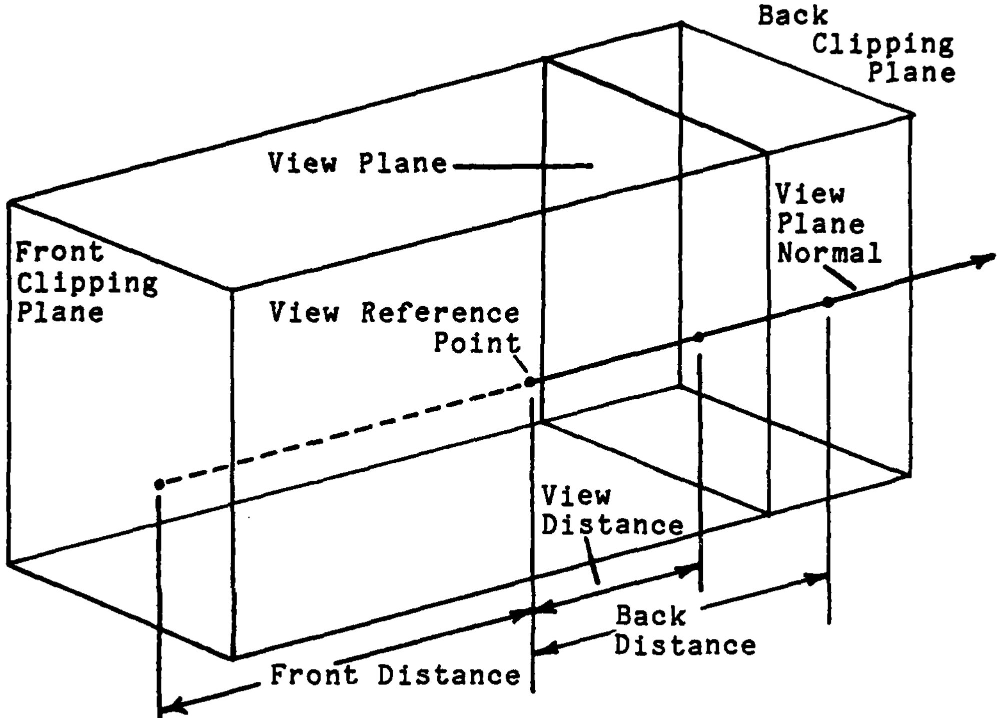
Set_View_Plane_Distance(D) - locates the view plane a distance D (in world units) from the view reference point measured along the view plane normal.

Set_View_Up(DX,DY,DZ) - DX, DY, and DZ define a unit vector in world coordinates relative to the view reference point. The view up vector is parallel projected onto the view plane in the direction of the view plane normal to determine the vertical view plane axis.

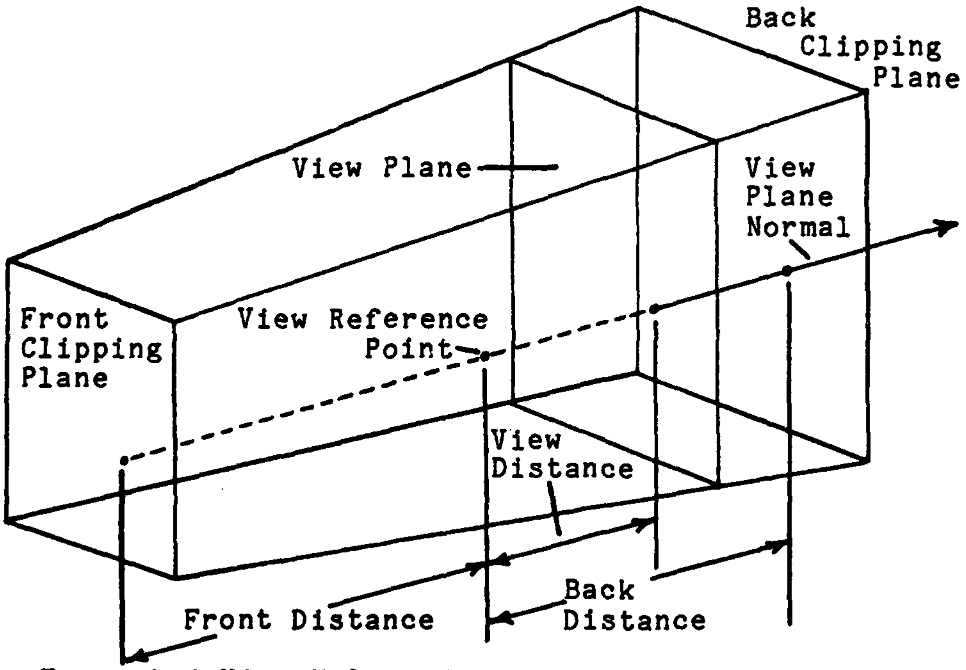
Set_Projection(Type,DX,DY,DZ) - the parameter Type determines whether the projection onto the view plane is to be parallel or perspective. The parameters DX,DY, and DZ specify the projection point in world coordinates relative to the view reference point. A line drawn from the projection point to the reference point determines the direction of parallel projection. For perspective projections, the projection point is the center of projection.

Once the 3-D image has been projected onto the view plane it can be treated just like a 2-D image. This means that a window must be specified to determine the area of the view plane that is to be mapped to the viewport. The same function used to set the window parameters for 2-D operation can be used for 3-D. The only difference is that the parameters, while still being in world units, will specify a window with respect to some origin on the view plane rather than the origin of the world coordinate system. This difference should be taken care of internally by the graphics system and hidden from the user.

Just as in 2-D graphics, clipping is also a part of 3-D graphics. Since the images are three-dimensional, clipping will be done using a volume rather than a two-dimensional rectangular area. The shape of the clipping volume will depend on the type of projection specified: a parallel projection will require a parallelepiped volume (Figure 11a) and a perspective projection will require a pyramid (Figure 11b). A limit is normally put on the length of the clipping volume by specifying front and back planes. The front plane will prevent, for example, any part of the image that is behind the viewer from being projected onto the view plane. These planes are also shown in Figure 11.



a. Truncated View Volume for Parallel Projection



b. Truncated View Volume for Perspective Projection

Figure 11. Clipping Volumes [Ref 20:II-54]

Set_View_Depth(Front,Back) - specifies the distance for the front and back clipping planes from the view reference point as measured along the view plane normal.

The sides of the view volumes are automatically determined by the size of the window in the view plane and do not need to be specified by the user program. Also, as in 2-D, there may be times when it is desirable to turn off clipping, so there could be a function included to turn clipping on and off.

Summary

This chapter has provided a brief discussion of the basic principles involved in computer graphics. The different function sets that are necessary for a simple graphics package were identified. The advantages of device-independence and how to include this feature in a graphics package were also covered.

III Graphics System Requirements

This chapter presents the general requirements for a device-indepedent graphics system using the function sets described in Chapter II. The advantages of having a graphics package that is both device-independent and host-independent were discussed in the previous two chapters. The ACM Core System that was mentioned in Chapter I is a proposal for a device-independent graphics standard and will be covered in some detail in the next chapter.

Any graphics system standard, just because of the fact that it is a standard, has to provide a high degree of independence. When specifying the requirements for a standard graphics system, a compromise must be made between the amount of capability provided and the level of independence acheived. As more and more functions are added to the package, there is an increased risk of the package becoming tailored too closely to a particular application or device, thus limiting its widespread use. On the other hand, the package should not be so basic that it is not powerful enough to be of much use in any but the simplest of applications.

Figure 12 shows a basic block diagram for a graphics system. It illustrates the division of the graphics functions into those concerned with describing an image, the

modeling functions, and those functions dealing with actual picture generation: primitives, segmentation, and viewing transformations.

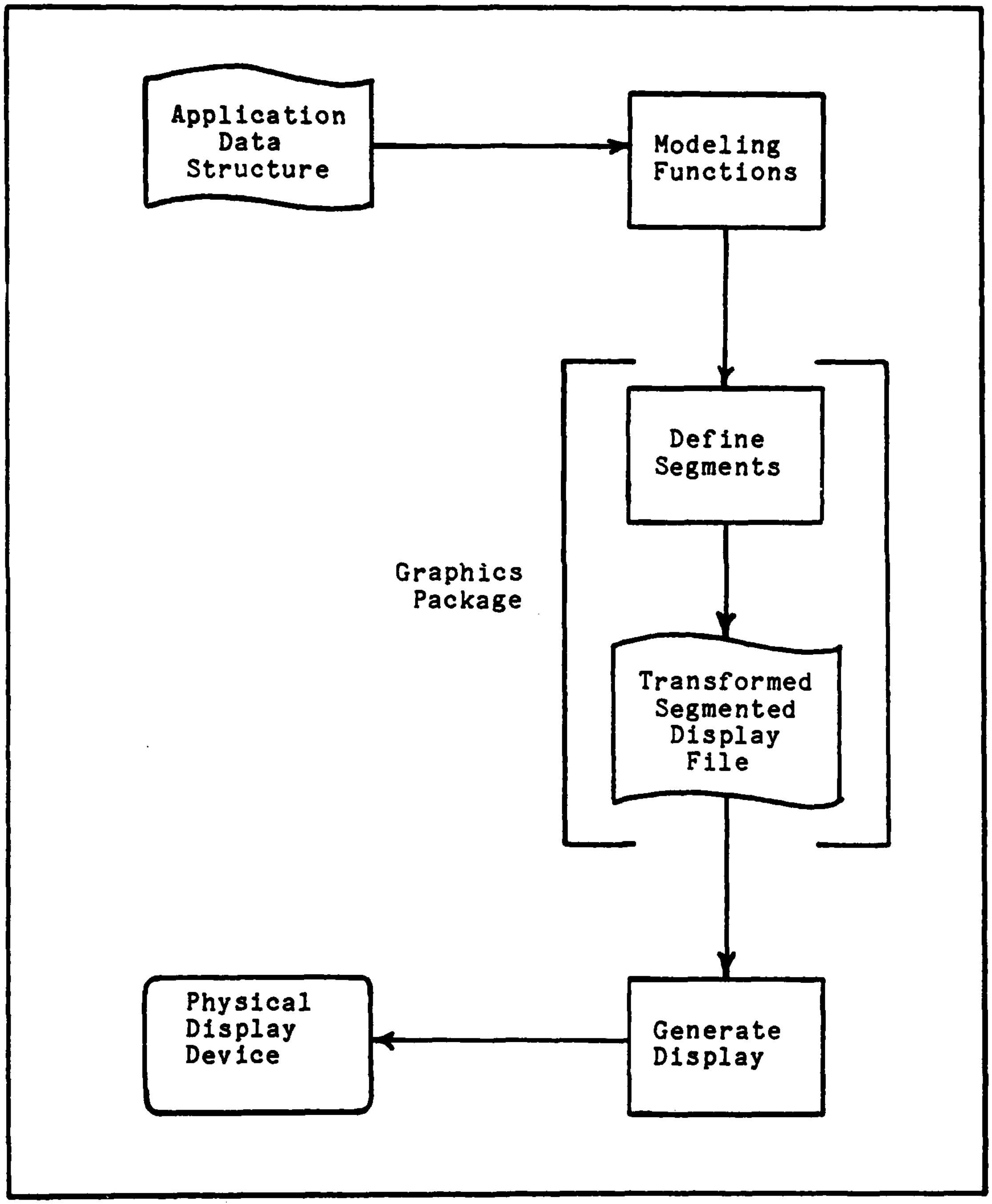


Figure 12. A Basic Graphics System [Ref 12:433]

Modeling Functions. There is a wide range of transformations involved in constructing models for images and many are application dependent. These can be non-standard or non-linear and not easily generalized into a common operation such as a 4x4 matrix multiplication. This is why the modeling functions should be separated from the basic picture-generation routines. The modeling functions required for a certain application can be put into a specialized set of library routines that will produce a standard world-coordinate model of the image. The general-purpose graphics package can then be used to generate the output display.

The instance transformation described in Chapter II is a modeling function, but it is the same scale-rotate-translate geometric transformation as used for image transformations. As will be shown shortly, the same type transformation is also used for 3-D graphics when converting into the view point coordinate system. Since the instance transformation and the 3-D conversion are consecutive operations, and are both matrix operations, the two conversions can be concatenated to create one compositie operation. This is a great saving in the amount of computation required. In addition, the instance transformation is one of the most basic and widely used of the modeling functions. For these reasons, the instance transformation should be made a part of a standard graphics

package, rather than being left for each user to implement as part of his modeling library functions.

The result of any of the modeling functions is to transform the coordinates of the graphics primitives from a user coordinate system into the world coordinate system. The user system will use world units so this is a world-to-world transformation. Once the parameters of the graphics primitives have been specified with respect to the origin of the world system, the general-purpose graphics package can then take over the processing of the graphics functions.

Output Functions. Clipping, performing the viewing transformation, and handling picture segments are the main operations of the graphics package. The sequence of these operations is shown in Figures 13a & 13b. For two-dimensional graphics, the world-coordinate images are clipped to the window prior to the viewing transformation. Three-dimensional graphics is more complicated, however, because of the use of the view point and the view plane, as discussed in the previous chapter. Since 3-D images must be clipped to a view volume and then projected onto a view plane, where both actions are determined by the view point, it simplifies matters if each stage of the operation uses its own coordinate system. All coordinates will still be expressed in world units, however.

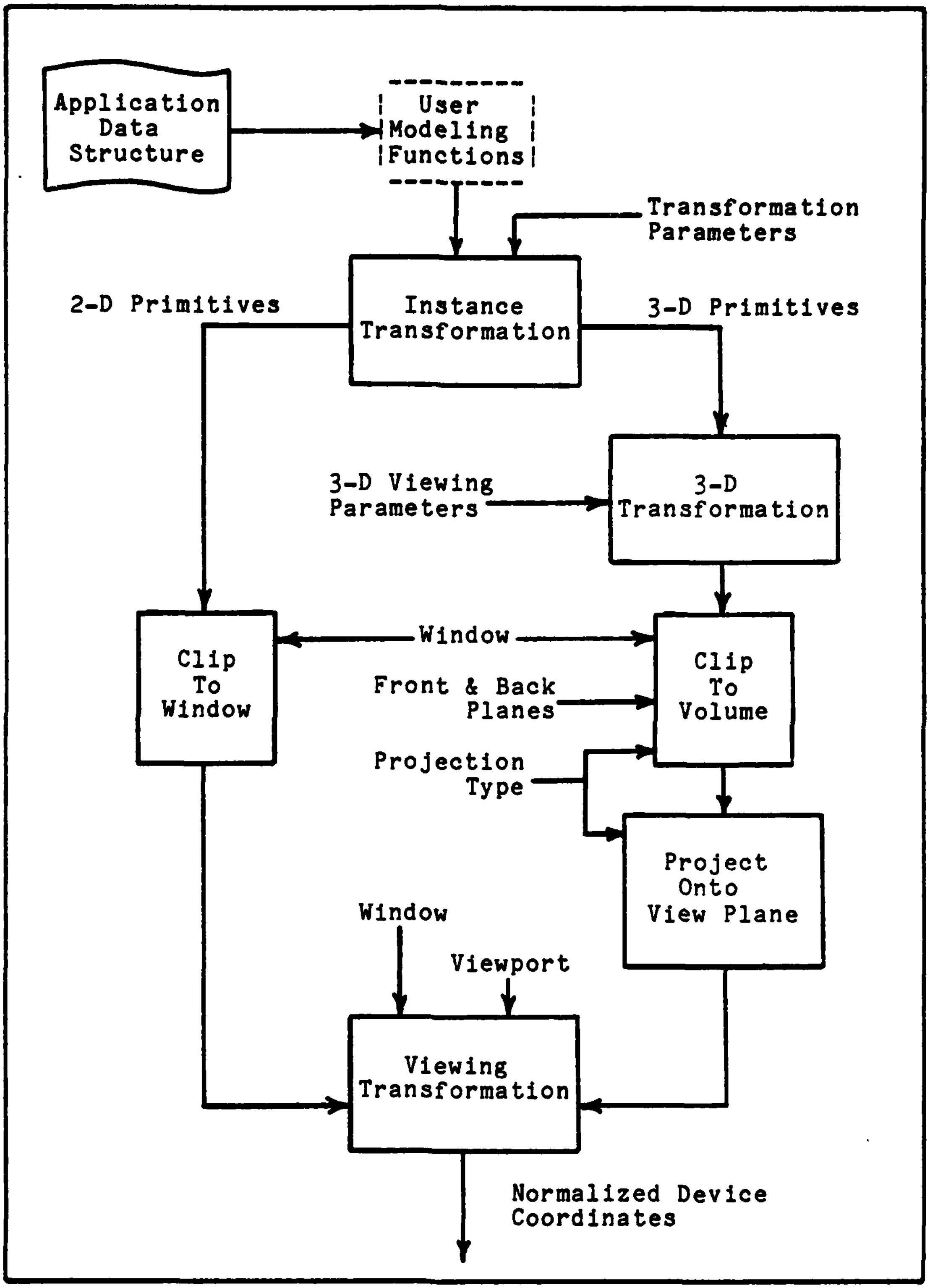


Figure 13a. Basic Graphics System
-Clipping and Transformation-

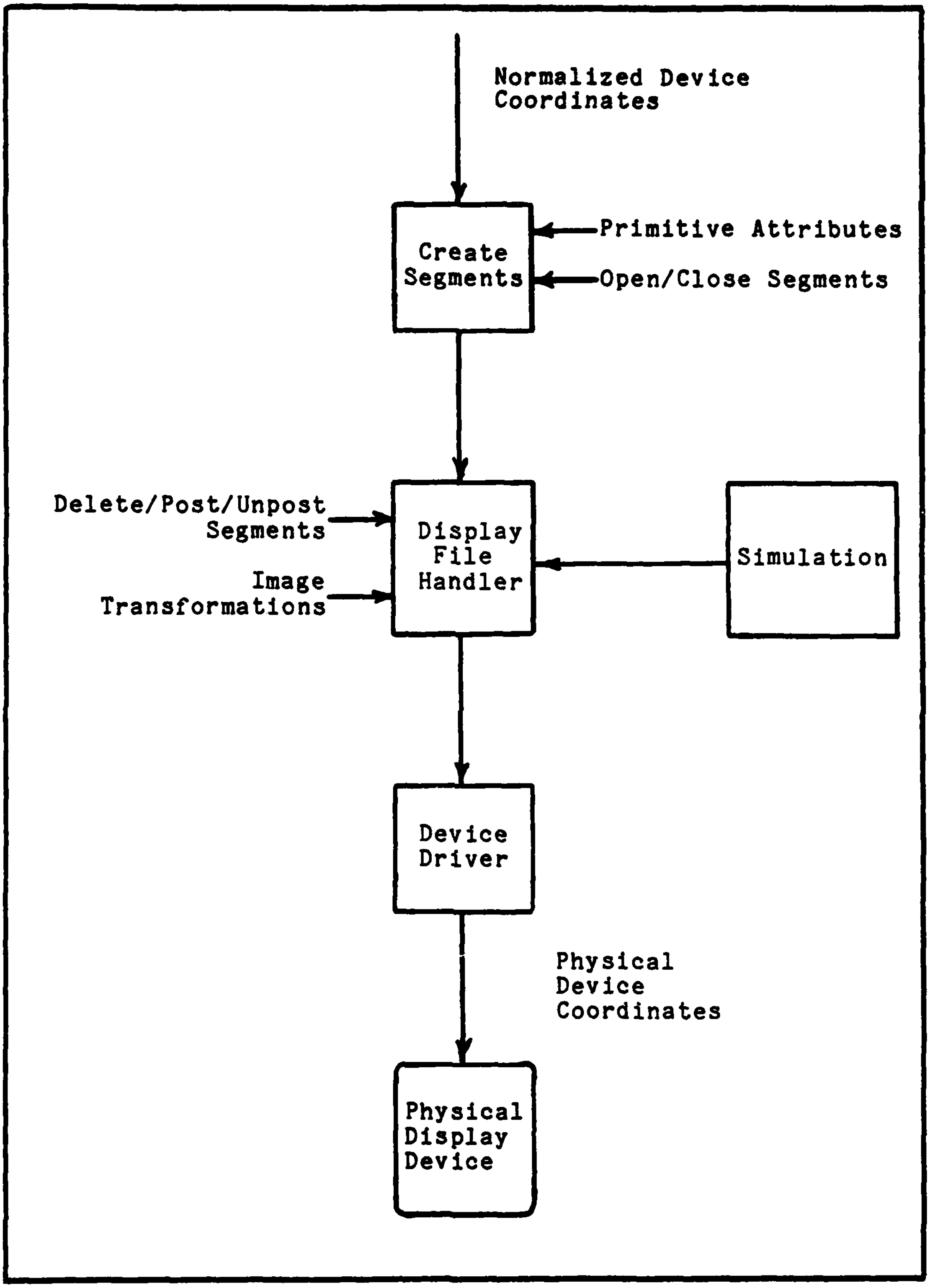


Figure 13b. Basic Graphics System
-Segmentation and Display-

The view point is often called the "eye" point because it is the location from which the 3-D image is being viewed. It will be the origin for the "eye" coordinate system. Likewise, a point on the view plane will be specified as the origin of the view plane coordinate system. The location of this point should be determined automatically as a function of the 3-D viewing parameters and not left for the user to specify.

A simple geometric matrix operation can be used to transform primitive coordinates from the world to the eye system. As mentioned earlier, this operation can be concatenated with the instance transformation to convert from the user to the eye system in one operation. The transformed image will then be clipped to the view volume and, if visible, projected onto the view plane. The viewing transformation can then be performed just as in 2-D to transform into normalized device coordinates.

At this point, the graphic primitives are placed into segments and the current values of the primitive attributes are assigned to the primitives. The completed segments are then placed into a display file from which the device drivers produce the output drawing on the corresponding device. The display file handler provides the interface between the device-independent graphics package and the device-dependent driver routines. As has been pointed out previously, this provides one of the major benefits of a

standard graphics package: the ability to use the package with a wide range of graphics devices simply by changing the device driver that is used with the package.

Device Drivers. Since graphics devices have such a wide range of capabilities, a decision must be made as to the level of interface between the graphics package and the device drivers. That is, at what level should the commands be that are sent to the device drivers? (The number of primitive functions provided also plays a part in this decision.) If the graphics package remains at a very basic level, then the high capability of some displays will not be used. On the other hand, higher-level fuctions that make use of high-capability displays will require the drivers to simulate these functions for devices that are not as capable. Requiring the simulation of many of these high-level functions can make the drivers quite complex and will also result in some duplication of code from one driver to another. To get around this, and make the drivers simpler, the simulation can be done in the graphics package itself. When initialized, the drivers can inform the graphics system as to the capabilities of the device. The graphics system can then provide the necessary simulation when needed by a device, but still not prevent full utilization of a high-capability device.

Input Functions. The main use of input functions is to transmit input data from the graphics user to the

application program, where a response is generated using the output functions. Device-dependent input routines are provided as part of the device drivers. The interface between the input and output parts of the graphics package is limited. Areas in which the input and output sections might interface would be initialization, where certain input parameters are set up along with the rest of the graphics system; echoing, where acknowledgement of the operator's actions is automatically sent to the output device; and picking, where the input system must associate a selected screen position with a particular picture segment. For details on the design of an input graphics package, refer to Capt Curling's thesis [Ref 3].

Utility Functions. These are the functions that support the input and output functions and provide the housekeeping chores for the graphics package. They deal primarily with system parameters in common areas by either initializing them to default values or retreiving their current values through an inquiry function. For complete flexibility, there should be an inquiry function for every system parameter. In most cases these functions need to just retrieve parameter values directly from the common storage area, but some parameters will require application of a reverse transform to get the parameter into the proper coordinate system if that parameter is normally transformed prior to being stored. As mentioned in Chapter II, utility

functions should also be provided to initialize the graphics system and the input and output devices.

Summary

This chapter has presented the general requirements for a device-independent graphics package. Topics covered included modeling functions, input and output functions, device drivers, and utility functions. The requirements are based on the basic graphics principles that were described in Chapter II.

IV An Overview of the Core Graphics Standard

Now that the general requirements for a device-independent graphics package have been established, this chapter provides a more detailed look at graphics requirements with an examination of a specific proposal for a graphics standard.

Although there is no officially adopted national or international standard for computer graphics, the ACM Core proposal that was discussed in Chapter I represents the best attempt made so far to create a standard. While it is only a proposal and has not been officially adopted, it is being studied by standards organizations as the basis for a future graphics standard. Until such a standard is eventually adopted, the ACM Core proposal provides the most logical choice on which to base the design of a graphics package. Therefore, the ACM Core proposal will be the basic requirements document for this effort.

The Core Graphics Standard is documented in the "Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH" [Ref 20]. It covers changes to the Core System of 1977 [Ref 19], general methodology of the design, the actual proposed standards, raster extensions to the standards, the GSPC Metafile proposal, and a report on distributed graphics systems. This overview will cover the methodology, proposed standards, and raster extensions.

Methodology

The Core System was designed mainly with program portability in mind. This is defined as "the ability to transport graphics applications from one installation to another with minimal program changes" [Ref 20:II-1]. As pointed out in the report, transportation of programs can involve three levels of program changes:

1. No source program changes at all.

2. Modifications that are purely editorial and do not involve changing the program structure. An example would be substitution of the instruction LINE for DRAWTO.

3. Modifications to the structure of the program itself.

While some programs may be transported without any changes, the majority of programs will require that at least some changes be made. This is true even of programs written in a "standard" programming language such as FORTRAN. Implementations of FORTRAN can vary from one installation to another so it is not unusual to have to slightly modify non-graphic programs when they are transported between installations. The same will also be true of most graphics programs.

Programs requiring modifications to their structure are the most difficult to try to standardize. These programs are generally highly interactive and written to take advantage of device-specific hardware functions. In fact, the wide diversity of input and output functions available with current graphics hardware is one of the biggest problems in standards design and can be considered one of the disadvantages of standardization. If the standard does not contain enough features, then there is the danger of not being able to effectively use some capabilities of high performance displays. On the other hand, if the standard contains too many features, software simulations will be required with less-capable devices. While not being able to eliminate structure changes when transporting programs, the Core System designers attempted to at least preserve the program's basic dialogue between operator and machine.

The Core System makes use of the following basic concepts:

1. The separation of input and output functions.

2. The minimization of the differences between producing output on a plotter and on an interactive display.

3. The concept of two coordinate systems: the world coordinate system in which the picture for display is constructed, and the device

coordinate system in which data to be displayed is represented.

4. The concept of a display file containing device coordinate information; used by all but the least interactive of graphics systems.

5. The notion of mutually independent display file segments, each of which can be modified as a unit.

6. The provision of functions to transform world-coordinate data into device coordinates by invoking a viewing operation.

These are many of the same concepts that were discussed in Chapters II and III. As a brief review, the general sequence of graphics operations is as follows:

> The application program creates a definition of the objects to be displayed in the world coordinate system. This definition then passes through a viewing operation which creates a representation in normalized device coordinates. The viewing operation will normally include a clipping operation to define what portion of the world coordinate system is to be displayed. The normalized coordinate definition is stored in a display file so that a refresh display may be maintained and so

that individual segments of the picture may be changed without regenerating the entire picture. For interactive programs, input from the operator will cause the application program to compute new data values and/or change the picture being displayed.

An issue that the GSPC had to consider in the design of the Core System was what to include in the standard and what to leave out. This is a point that cannot be easily agreed upon; there will always be arguments that the standard contains too many functions or does not contain enough. In deciding on a cutoff point for what to include in a set of standards, the designers classified graphics functions into two main groups, *viewing and modeling*. *They put everything* strictly concerned with picture generation into the viewing system, and anything relating to the construction and manipulation of the objects being displayed into the modeling system. The modeling system is more application-dependent, and is something the designers feel requires more work in order to develop a basic modeling system that can interface with the graphics (viewing) system. This separation of the graphics functions into two groups was discussed in Chapter III.

Functional Specification

As has been mentioned several times, the main purpose of a graphics standard is to provide program portability. This is done by shielding the application programmer from the specific hardware characteristics of the system by performing all input and output with respect to a standardized group of logical devices. The application program specifies the graphical objects in device-independent world coordinates. The display of these objects is specified on a logical view surface in normalized device coordinates. In addition, logical input devices can be specified by the applications programmer without concern for the specific hardware devices. Mapping the logical view surface and the logical input devices to the actual physical devices is handled by the Core System.

The Core System's functional capabilities have been divided into the following categories: output primitives, segmentation, attributes, viewing transformations, input primitives, and control. These match closely with the categories of graphics functions that were described in Chapter II.

Output Primitives. Objects in two- or three-dimensional world coordiates are described as combinations of output primitive functions. They consist of the following:

MOVES

LINES

POLYLINES (a connected sequence of lines)

MARKERS (to designate points on plots)

TEXT STRINGS

A current position is used as the starting point for lines and text strings. The output primitives' position can be specified in either absolute or relative coordinates.

Segmentation. An application program describes an object by creating (opening) a segment, invoking output primitive functions, and then closing the segment. Each segment represents an image of an object in the displayed picture. There are two types of segments in the Core System: retained and temporary. Retained segments, which are named, can be deleted, renamed, and have their attributes modified. They are retained until explicitly deleted. Temporary segments are displayed only once and no record is kept of their contents. Segments cannot contain references to other segments; thus, picture hierarchies and copies of segments cannot be created.

Attributes. Attributes define characteristics of retained segments and output primitives [Ref 20:II-30,II-31,II-36].

1. Segment Attributes:

VISIBILITY - determines whether the segment is actually displayed on the output device.

DETECTABILITY - determines whether the segment can be "seen" by a light pen or other pick device.

HIGHLIGHTING - when enabled, calls the operator's attention to the segment by blinking or intensifying it.

IMAGE_TRANSFORMATION - indicates how the image of a retained segment is scaled, rotated, and translated.

IMAGE_TRANSFORMATION_TYPE - determines what image transformations are valid for the retained segment.

Except for IMAGE_TRANSFORMATION_TYPE, the above segment attributes are dynamic - they can be changed either while a segment is being constructed or at a later time after it has been closed.

2. Primitive Attributes:

COLOR	- indicates the color of the image of a visible primitive.
INTENSITY	- indicates the relative brightness of the image of a visible primitive.
LINESTYLE	- indicates the style of the image of a visible line (e.g., solid, dashed).
LINEWIDTH	- indicates the relative width of the image of a visible line.
PEN	- indicates the "pen" used to distinguish the image of a visible primitive.
FONT	- indicates the style of a visible text character (e.g., Roman, Gothic, italic).
CHARSIZE	- indicates the desired size, in world coordinate units, of a text character.

CHARPLANE	- indicates the orientation in the world coordinate system of the plane on which the text characters appear.
CHARUP	- indicates the principal up direction in the plane on which the text characters appear.
CHARPATH	- indicates the string direction (right, left, up, down) within the plane on which the characters appear.
CHARSPACE	- indicates the additional spacing between adjacent character boxes in a string.
CHARJUST	- indicates the mode of string justification for both horizontal (left, center, right, off) and vertical (top, center, bottom, off) directions.

CHARPRECISION - indicates the precision (string, character, stroke) of the appearance of a TEXT primitive.

MARKER_SYMBOL - indicates the symbol used to denote the position of a visible marker primitive.

PICK_ID - indicates the name of a primitive which is returned for use by the application program whenever the primitive is selected by the operator using the PICK input device.

Once an output primitive is created, its attributes can be changed only by deleting the segment that contains the primitive and then respecifying a new segment.

Viewing Transformations. These transformations are used to map an object described in world coordinates onto the logical view surface given in normalized device coordinates. The boundaries for these areas are the window and the viewport, respectively. The window is used to clip two-dimensional objects and to determine the window-to-viewport mapping. For three-dimensional graphics, the window is specified in an arbitrary view plane, which is

then mapped into the viewport. The 3-D objects can be clipped against a 3-D solid volume: a pyramid for perspective projections and a solid rectangle for parallel projections. The image of an object, whether 2-D or 3-D, can be translated, scaled, and rotated after the image has been created. Again, these functions are the same as those described for the general graphics requirements in Chapter III.

Input Primitives. Input primitives are associated with two logical input devices - event and sampled. The event devices are those that signal events to the user application program and are generated by the operator. The sampled devices are those which the application program polls for information. There are several classes of devices associated with these two device types [Ref 20:II-71 - II-73]. They are:

1. Event Devices

PICK - provides the segment name of the output primitive that was "picked" by the operator.

KEYBOARD - provides character string input.

BUTTON - provides the operator with a method of specifying application program functions.

STROKE - allows the operator to draw a line on the view surface by specifying a series of positions on that surface.

2. Sampled Devices

LOCATOR - provides coordinate information in normalized device coordinates.

VALUATOR - provides a scalar input to the application program within a range of values specified by the user.

Each of these logical devices must be implemented either physically or by simulation. They also must be initialized and enabled in order to be used. The input primitives are the routines that accomplish the graphics input functions.

Control. Functions are provided for initializing and terminating the Core System. The general operating environment of the Core System can be controlled by turning clipping on or off, selecting one of several display devices for output, setting the standard default values for segment and output primitive attributes, and establishing eror handling procedures. In addition, inquiry of Core System and device capabilities and current state is also provided.

Levels of Implementation. The Core System is meant to be used with a wide range of graphics devices, from simple plotters to highly interactive refresh displays. Plotter users do not require as many of the Core System features as would the interactive user. The existence of one all-encompassing Core System would either force the plotter user to have the software overhead of a system much larger than needed, or to select an arbitrary implementation of only those features that were needed. This latter option would result in a number of non-standard implementations, thereby defeating the goal of program portability. To get around this problem, three classes of upwards compatible levels have been specified by the Core System designers: one class for output, one class for input, and one class concerned with the dimensionality of the world coordinate space. These classes are described below [Ref 20:II-10 - II-12]:

1. Output Levels

   Level 1: Basic Output

   This level supports the full set of output primitives, primitive attributes, and viewing operations that are appropriate to the dimension level selected. Temporary segments must be used.

Level 2: Buffered Output

This level allows the use of retained segments. The retained segment attributes of highlighting and visibility are supported and all retained segment operations are supported. Detectability is only supported if Input Levels 2 or 3 are also implemented.

Level 3: Dynamic Output

This level provides the complete set of picture presentation capabilities, including image transformations. In order for an implementation to take advantage of those image transformations supported by hardware, without providing software simulation of the remaining transformations, three sublevels have been defined:

Level 3A: 2-D Translation

Level 3B: 2-D Scale, Rotation, Translation

Level 3C: 3-D Scale, Rotation, Translation

2. Input Levels

Level 1: No Input

No input primitives are supported. Therefore, this level is appropriate for output-only applications.

Level 2: Synchronous Input

This level supports applications which have no requirements for asynchronous interaction. No explicit enabling or disabling and no device associations are either required or allowed. All input device classes are supported, but PICK is supported only if Output Levels 2 or 3 are implemented.

Level 3: Complete Input

This level provides full support for all input functions defined by the Core System.

3. Dimension Levels

Level 1: Two-dimensional (2-D)

This level provides only 2-D operations.

Level 2: Three-dimensional (3-D)

This level provides both 2-D and 3-D operations.

Raster Extensions

These extensions were provided to update the original 1977 Core Report, which did not address raster devices. Significant advances had been made in the area of raster hardware that brought the price down and led to wider applications of raster equipment. The high interest in raster graphics resulted in the inclusion of raster functions in the updated 1979 report. To allow the Core System to be used for raster graphics, the extensions provide functions for:

1. Filled-in polygonal areas.
2. Extensive color and intensity specifications.
3. Display of patterns (pixel arrays) which are computer generated.

Polygonal Areas. In addition to drawing lines, text, and markers, raster displays have the capability to define and fill the interiors of plane polygons. The interior of a polygon image may be filled in one of three different ways, depending on the value of the primitive attribute POLYGON_INTERIOR_STYLE [Ref 20:III-5]:

"Plain" - all of the interior pixels are assigned the same index value, as specified by the attribute FILL_INDEX.

"Shaded" - the interior of the polygon image is filled with pixel values that are the result of interpolating the index values of the vertices of the polygon.

"Patterned" - each interior pixel is assigned an index value out of an array of index values that contain the specified pattern.

The edges of the polygon image may be displayed in one of two ways, depending on the value of the primitive attribute POLYGON_EDGE_STYLE:

"Solid-Line" - the edges of the polygon are displayed as solid lines.

"Interior" - the edges of the polygon are displayed as the continuation of the interior.

<u>Color and Intensity</u>. Two different models of color are supported by the Core System [Ref 20:III-6 - III-7]. One is the widely-used RGB (red, green, blue) color model. The other is the HLS (hue, lightness, saturation) color model. The user selects which model is to be used via the SET_COLOR_MODEL function. This is done before any view-surface has been initialized. Once selected, the model cannot be changed. Color and intensity are associated with output primitives by index value and not directly. Each

index value is used to look up an associated color or intensity specification in a look-up table. Each view surface has a color and itensity look-up table associated with it. Those devices which do not have a hardware look-up table are provided with one in software. A full set of functions is provided to define and inquire about the contents of the look-up table associated with a view surface. Devices can be split into two classes, depending on how changes to their look-up tables are handled [Ref 20:III-8]:

"Sequential" - a change in the contents of the look-up table can affect only the images of primitives drawn after the change.

"Retroactive" - a change in the contents of the look-up table can affect the images of all primitives.

<u>Pixel Arrays</u>. A pixel array is a two-dimensional rectangular array of index values which is used to fill the image of polygons which have the "patterened" attribute. For the purpose of displaying, the pixel array is replicated in both the X and Y dimensions until each pixel of the polygon image has a corresponding pixel array element. The result of mapping a pixel array to a view surface is called a pattern.

Since raster graphics provides the capability to define surfaces by polygon filling, a problem arises when many segments are displayed at one time: one surface may "hide" another. Therefore, the removal of hidden surfaces must be addressed. The problem of hidden surfaces is application-dependent, so to account for the various user needs, the implementation of hidden surface removal has been divided into levels. The parameter HIDDENSURFACE is added to the current Core System level parameters of OUTLEVEL, INLEVEL, and DIMENSION. The HIDDENSURFACE levels are as follows [Ref 20:III-12]:

"None" - no software is provided for hidden surface removal. If the images of output primitives overlap on the view-surface, the result will be device-dependent.

"Temporal" - for all output primitives within a batch-of-updates (a list of primitives waiting to be displayed), the visibility of the images will depend on the order in which calls are made to the output primitive functions.

"Explicit" - this level is popularly known as 2 1/2 D. For all output primitives within a batch-of-updates, priority order is

determined by which Z-plane contains the primitives. For primitives within the same plane, temporal priority is used.

"Full" - this provides full 3-D hidden surface removal for all output primitives within a batch-of-updates.

Summary and Comments

Except for having a separate section devoted to attributes, the major functional areas into which the Core System is divided are the same as presented in Chapters II and III for a basic graphics package. In this respect, the Core System matches closely the Chapter III requirements. In fact, many of the individual functions are identical to those described in Chapter II. However, the Core System goes beyond those basic requirements in each functional category and provides a more versitile and useful standard package while still maintaining the needed device-independence.

For additional information on the Core System, the reader is referred to articles by Michener and Bergeron. In one article [Ref 10], Michener and Foley provide a background discussion of the issues considered in the design of the Core System, various alternatives that were looked at, and the reasons for selecting the final design. In a

second article [Ref 9], Michener and Van Dam provide a brief overview of the major features of the Core System. Using many examples, Bergeron [Ref 1] presents a good tutorial on the principle concepts of interactive graphics programming using the Core System.

The functional capabilities presented by the Core System design are too numerous for one individual to implement in the short time allotted to a thesis. The goal of a full implementation of the Core System on the AFIT VAX must be the result of the work of several individuals over a long period of time. The design work of Capt Curling and the use of an existing output system implementation are major steps towards reaching that goal.

The goal of this effort is to integrate the input design of Capt Curling with an existing output system to provide a partial implementation of the Core System. This partial implementation will result in a package that provides the following Core System levels:

Output Level 3B	- Dynamic output providing 2-D scale, rotation, and translation.
Input Level 2	- Synchronous input.
Dimension Level 1	- Two-dimensional.
Hidden Surface Level 1	- "None".

V Implementations of the Core Standard

The initial task of this effort was to obtain an existing package of graphics output routines for implementation on the AFIT VAX. This chapter presents the results of the investigation of graphics packages that meet the requirements of the Core Standard.

A report by Gary Chappell [Ref 2], Chairman of the ACM/SIGGRAPH/GSPC Implementation Subgroup, provides a brief listing of some of the graphics system implementations that resemble the Core System. His report was assembled in the Fall of 1979 and published early in 1980. At the time, Mr. Chappell was aware of 27 implementations, but was only able to provide information on 18 of them. Portions of the survey results are listed in Table I.

The graphics systems mentioned in Chappell's report implement the Core at several different levels using a variety of graphics devices and host computers. Some of the graphics systems (Aydin Controls, Imlac, Vector General) are designed for specific graphics devices and can only be obtained by purchasing those devices; a few others (Control Data, Systems Design Group, Tektronix) are commercial packages that are for sale; and some (Grinnell College, Los Alamos Lab, University of Utah) provide only the lowest input and/or output levels of the Core Standard.

Table I

Core System Implementations [Ref 2:261-278]

Organization	Host Processors	Core Levels
Aydin Controls	PDP-11; VAX-11	IN-3;OUT-3B;2-D
Control Data	CYBER 170,70;CDC 6000	IN-2;OUT-2;3-D
DOD (NSA)	PDP-11/70	OUTPUT ONLY;2-D
G. W. Univ.	VAX-11/780	COMPLETE; 3-D
Grinnell College	PDP-11/70	IN-1;OUT-1;2-D
Hartford Grad Cen	PDP-11/60	OUTPUT ONLY;2-D
Imlac Corp.	PDP-10,11;PRIME;CYBER XEROX SIGMA 7;HP 2100	IN-3;OUT-3A;2-D
L. Livermore Lab	CDC 7600; STAR; CRAY	OUTPUT ONLY;3-D
Los Alamos Lab	CDC 6600,7600; CRAY	IN-1;OUT-1;2-D
Sys. Design Group	PDP-11; VAX-11/780	OUTPUT ONLY;2-D
Tektronix, Inc.	IBM 370;DEC-10;PDP-11 CDC;VAX;PRIME;HP3000	IN-2;OUT-1;3-D
Univ. of Alberta	PDP-11/60;AMDAHL 470	IN-?;OUT-?;3-D
Univ. of Colorado	CDC 6400; CYBER 175; PDP-11/70;AMDAHL 470; HONEYWELL 6000; VAX	IN-2;OUT-3;3-D
Univ. of Penn.	APPLE II	IN-2;OUT-2;3-D
Univ. of Utah	DEC-20; IBM 360/370; CDC 6000/7000/CYBER	IN-1;OUT-1;3-D
Corps of Eng.	CDC;HONEYWELL;PDP-11	OUTPUT ONLY;3-D
Vector General	PDP FAMILY	IN-3;OUT-?;3-D
W. Michigan Univ	PDP-10;HARRIS SLASH 7	IN-?;OUT-?;2-D

All of the systems except two are written in either FORTRAN '66 or FORTRAN '77, with nearly half also using assembly language for some portions of the code. The two language exceptions are the National Security Agency version, which uses 'C', and the University of Pennsylvania version, written in Pascal. Although the Lawrence Livermore package is written in LRLTRAN, a non-standard FORTRAN, a standard FORTRAN version is available.

Since his survey was undertaken so soon after the revised Core Standard [Ref 20] was published, all of the listed graphics systems had only the original 1977 Core Report [Ref 19] to use as a guideline. A similar survey that outlines the current status of these graphics packages and whether they have been updated to reflect the guidelines of the revised Core Standard could not be found. Nor has there been found a list of additional systems that have since been developed. The graphics packages that will be described in this chapter are those individual systems mentioned in Chappell's survey for which updated information was located.

George Washington University - GWCORE

A description of this graphics package appeared in the August 1981 issue of Computer Graphics [Ref 6]. James Foley, a professor at the University, was also a member of some of the GSPC subcommitties that worked on the Core

Standard design. He therefore has the knowledge and familiarity needed to implement the Core Standard in the manner in which it was intended. In fact, the design document for GWCORE claims that this is the first complete implementation of the proposed 1979 Core System [Ref 21:2].

The output part of GWCORE was released in September 1980. The complete integrated input/output system (input level 3, output level 3, dimension 3-D) was due to be released in the spring of 1981. The system is to include all raster extensions except hidden-surface removal.

GWCORE is written in DEC VAX-11 FORTRAN IV-Plus. A device driver for a Tektronix 4012 is written in FORTRAN and a RAMTEK 9400 device driver is written in C, although the driver for the latter device will only be released to those having a UNIX license.

The package contains 230 subroutines with 22000 lines of source code, of which approximately 35% are comments. A capability to handle polygons with 1500 vertices is provided. They estimate that an average-sized program will require no more than 450K bytes of memory.

GWCORE is divided into three main areas: the device-independent core, a device-independent to device-dependent interface (DI/DD), and device dependent drivers. This provides the advantage, as previously discussed in this document, of being able to support a new

device by simply writing a new device-dependent driver. When initialized, the driver reports which functions it actually supports. The Core System then simulates the capabilities not provided by the driver. This process greatly simplifies device driver implementation.

The DI/DD interface was kept as simple as possible. All parameters are passed in an array through a single entry point, along with an op-code and length to allow the driver to know what function to perform and how to interpret the information in the array. There are a total of 70 output-function op-codes and 35 input-function, but a simple output-only device may require as few as 20 of these.

Separate paths are used in the device-independent core area for 2-D and 3-D applications, which saves computation time for the 2-D functions. The clipped and transformed output primitives are placed into a communication packet (array) and sent to a dispatcher. The data in the packet is organized in the same format as is passed across the DI/DD interface. The dispatcher decides which packets should be sent to the currently selected device-driver, which should be stored for a new-frame action, and when the simulation routines should be called.

The source code and machine-readable documentation are available from GW University for $800. The software has also been distributed to the National Energy Software Center

(NESC) and is available to registered installations. Other installations must pay a subscription fee to NESC. The total cost for the subscription fee and the GWCORE package should be under $1000.

Grinnell College

The survey form in Chappell's report for this package stated that the subroutines were available only locally, but this graphics package did become available in the spring of 1981 [Ref 7]. The implemented level is also higher than the very basic input and output levels reflected in the survey, but it is still far short of a full Core Standard implementation. More detailed information on this package was not available. The implementation guide and language guides can be obtained only by ordering the software.

This graphics package is available in both ANSI FORTRAN and DEC BASIC-PLUS and is running on a PDP-11/70. Drivers are currently available for the following devices: Tektronix 4010 and 4025, VT100 with graphics board, DEC GT42 scope, CALCOMP plotter, DECwriter with graphics board, and Printronix line printer.

I spoke with Gene Herman at Grinnell and he was not very enthusiastic about their package. In fact, he said that the newsletter announcing the release of the system was done without his knowledge or approval. He was apologetic

about the graphics and said that although it satisfies some of the local needs on campus, it was not a very "robust" package and some outside users would more than likely be disappointed in its features. The package is available from Grinnell for $35, but it comes with no guarantees as to correctness and the college cannot provide any consultation or debugging help.

Lawrence Livermore Laboratory - GRAFCORE

Lawrence Livermore National Laboratory (LLNL) has many powerful computers, including four CDC 7600's, two CDC STAR's, and a CRAY-1 [Ref 18:3]. There is a wide variety of graphics needs at the facility, and an associated variety of graphics packages. In 1978 [Ref 8:3-5], the LLNL Computer Graphics Group had received many requests for changes and upgrades to these graphics packages. At the same time, they were under pressure from the Department of Energy to produce transportable software to minimize the expensive duplication of software that existed among the various national laboratories. This was also shortly after the 1977 Core Standard proposal was published. As a result of these developments, the Graphics Group decided to satisfy all of their needs by developing one major graphics package.

The graphics package that was developed (and is still being revised and expanded) is called GRAFLIB [Refs 8; 15; 17]. It has been implemented at four levels (not to be

confused with the Core System levels): BASELIB, GRAFCORE, Level A, and Level B. Each level builds on the functions of the subordinate levels. BASELIB is at the very lowest level and contains functions for performing host-dependent data manipulation.

GRAFCORE is made up of J and K function sets. The K functions are very basic non-user graphics routines, specific to LLNL and its graphics devices. A listing of the K-function names and the purpose of each is contained in Appendix B. The J functions are the actual implementation of the Core Standard. A listing of the J-function names and the corresponding Core System functions, if any, is contained in Appendix A.

Function sets F, G, and H make up the Level A part of GRAFLIB. F and G are Level A user routines and H functions are non-user support functions for F and G routines. Level A routines provide those functions that are found most commonly in graphics applications, such as mapping coordinate data, drawing a grid or curve, or rotating objects [Refs 8:22-230; 13].

The outermost GRAFLIB level is Level B. This level contains many "canned" graphics routines for entire specific applications [Ref 16]. For example, one Level B routine can be used to draw an entire graph, complete with grid and axis numbering, title, labels, and the plotted data.

The designers of GRAFLIB intended that programming be done using the highest level functions that would satisfy user needs. They envision most "casual" programmers using Level B routines only. Applications programmers would use Level B when possible, but would also frequently use Level A and GRAFCORE routines. Production programmers would probably work only at the Level A and GRAFCORE levels.

The J and K GRAFCORE functions (along with the Level A functions) follow a naming convention; each function name is six characters long and begins with the letter of the associated function set. Next is a letter representing a verb that indicates the action the function represents (i.e., X for execute, S for set, I for inquire, etc.). The following four letters are two two-letter nouns or adjectives that describe the action of the function. For example, the function JPPL2A is a J-level routine (J) that plots (P) a polyline (PL) in two dimensions using absolute coordinates (2A). There are approximately 243 J-level functions and 210 K-level functions, although 103 of the J functions have not been implemeted [Ref 18:22-304]. Most of the unimplemented functions are: input functions, which GRAFCORE does not as yet support; many of the inquiry functions; and several multi-level functions. Multi-level functions are groups of fuctions that perform the same task but differ only in the number or type of arguments. An example would be the functions for plotting lines. There

are four separate LINE functions: 2-D, using either absolute or relative coordinates; and 3-D, using either absolute or relative coordinates. In this instance, only one of the four functions has been implemeted: the function for plotting lines in 3-D using absolute coordinates.

The SIGGRAPH Core Proposal figured strongly in the GRAFCORE design but it was not a strict requirement [Ref 18:3]. As a result, there is not a one-to-one correspondence between all the Core functions and the GRAFCORE J-level functions. GRAFCORE does follow, however, the main functional organization as discussed in previous chapters: output primitives, segmentation, attributes, viewing transformations, input primitives, and con rol. The comparison between the two function sets can be divided into three separate groups. In the first group are those J-level functions that correspond directly with Core System functions. Although all of these functions have not been implemented, the fact that they have all been named would indicate that there are plans for implementation sometime in the future. The second group contains all those J-level functions for which there are no corresponding Core System functions. Some examples of these would be axis and grid drawing functions, line buffering routines, and direct mode functions. Core System functions for which there are no corresponding GRAFCORE functions make up the third group. Some examples are functions for setting and inquiring

AD-A115 499   AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
CONTINUED DEVELOPMENT AND IMPLEMENTATION OF A STANDARD GRAPHICS--ETC(U)
DEC 81 P B TARBELL
UNCLASSIFIED   AFIT/GE/EE/81D-58   NL

2 OF 2
A
115499

END
DATE
FILMED
7-82
DTIC

retained segment dynamic attributes, synchronous input functions, and input inquiry functions. A listing of these three groupings can be found in Appendix A.

GRAFCORE operates in two modes: segment and direct. Segment mode is the implementation of the Core Standard, allowing picture parts to be stored in segments and transformed and plotted any number of times. Direct mode is not part of the Core Standard. This allows pictures to be displayed quickly using just one device at a time. The picture elements are not stored, and to send the same picture to another device requires that the picture be recomputed each time [Ref 14:1-2].

GRAFLIB/GRAFCORE is written in LRLTRAN, which is a special version of FORTRAN used at LLNL. However, standard FORTRAN source files may be obtained using a LLNL pre-compiler called PRECOMP. The standard method for calling one of the J functions from a user program is to use the following format:

```
IF(routine name(arguments))err,,err
```

The function will return a positive number if an error has occurred, and zero otherwise. In the future, this may be modified so that a negative number is returned for a warning error and a positive number for a fatal error.

There are three graphics devices for which GRAFLIB was initially designed, and there are many J and K functions written specifically for these devices. These devices are:

1. Television Monitor Display System (TMDS). This is a black and white raster monitor with a 512 by 512 resolution.

2. Information International Film Recorder (FR80). This is a vector device that plots graphics data on several types of high resolution film. The FR80 has a resolution of 16,384 by 16,384.

3. Honeywell Non-Impact Printer (NIP). This printer produces hardcopy output on three sizes of paper. Although it accepts vector graphics files, it works as a raster device.

Since this is a device-independent graphics package, many other graphics devices can be supported. Each device requires a separate device driver that must be implemented using the proper function names and programming conventions to correctly interface with the GRAFCORE package.

In addition to the three large mainframe types, GRAFLIB has also been implemented on a VAX 11/780 at LLNL which is using a Tektronix storage tube display for graphics output. This information is fairly recent and was, therefore, not reflected in Chappell's survey.

The survey states that this graphics package is available, but does not give any further details. Since the development was sponsored by the U. S. Department of Energy, there was a good chance that the software could be obtained by AFIT at no cost. This turned out to be the case, and to our advantage, Major Michael Wirth, of the AFIT Math Department, has a good working relationship with the Computer Graphics Group at LLNL and, in fact, has access to the LLNL computer network. As this thesis project was getting started, Major Wirth was already making plans to obtain a copy of GRAFCORE for use at AFIT.

U.S. Army Corps of Engineers - GCS

Information on the Army's Graphics Compatibility System (GCS) was obtained through the National Technical Information Service (NTIS) [Ref 4]. The information was not detailed and the Core Standard was not mentioned, although the host- and device-independent features of GCS were highlighted. According to the Chappell survey, which may be outdated, only output functions have been implemented.

GCS is written in FORTRAN and contains approximately 60,000 source code statements. Some high-level composite routines are provided to perform complex tasks such as the preparation of complete graphs and histograms. These sound similar to the Level B routines in GRAFLIB.

The graphics system is currently running on CDC, Harris, Honeywell, IBM, and UNIVAC machines (no model numbers supplied). Graphics output devices that have been supported include Tektronix 4010, 4014, 4027, 4622, 4633; CALCOMP plotter; Ramtek (again, no model number); and some unspecified alphanumeric terminals.

There are two versions of GCS, a two-dimension version and a three-dimension version. The two-dimension version is described as "the older, less sophisticated version". The two versions are available separately, each for a first-year lease of $1500, $1000 for subsequent years.

Selection of a Graphics Package

The first task to be undertaken for this project, as presented in Chapter I, was to obtain an existing output-function graphics package and get it running on the AFIT VAX. The ideal system to use for this effort would be one that was originally operating on a VAX, that meets the output level and dimensional requirements specified at the end of Chapter IV, and that can be obtained at little or no cost to the government.

Early in the investigation, GCS and GRAFCORE were the only two Core Standard graphics systems for which current information was available. GCS was not very desirable because of the cost involved and the fact it was not

operating on a VAX, but GRAFCORE met all of the stated requirements. (Although GRAFCORE had not been originally designed for a VAX, a version had been created that was operating on a VAX at LLNL.) Since plans were already being made to obtain that system when this effort was started, there was a strong case to be made in favor of using GRAFCORE.

However, the search continued for other graphics packages to see what types of systems were available. The Grinnell College package did not seem to provide any improvement over the capabilities of GRAFCORE, and in addition, was not operating on a VAX. The fourth graphics package, GWCORE, meets and exceeds the above requirements, except possibly for cost, and certainly appears to be an improvement over GRAFCORE. Unfortunately, the details on this system were not obtained until GRAFCORE had already been obtained.

GWCORE provides a full Core System implementation, plus a Tektronix device driver, so its use would have almost instantly resulted in a nearly complete realization of the goals of this effort. The only remaining task would have been to develop a device driver for the ISC 8001G color raster terminal.

GRAFCORE, on the other hand, is currently an output-only package at the 2-D level. It contains

provisions for future expansion, but these plans do not appear to include the complete Core implementation. Thus, the use of GRAFCORE would require the continued design and implementation of the input routines and both device drivers that had been the ultimate goals of this project.

Arguments can be made for the use of either package. GWCORE would save a lot of time and effort and allow further work to be done at the graphics application program level. But this would be at the loss of the learning experience that can be gained through the design and implementation efforts required to obtain a more capable package based on GRAFCORE. Just the opposite is true for GRAFCORE: it provides the educational experience, but would also be a duplication of effort to implement functions that are already available in the GWCORE package.

If details on GWCORE had been available when this investigation was first beginning, the decision might still have been made to use GRAFCORE simply because of the fact it was already in the process of being obtained, as mentioned earlier. If, however, the problems that were encountered in trying to implement GRAFCORE could have also been known at that time, an effort would have been made to use GWCORE right from the start.

The biggest advantage of the GWCORE package, in my opinion, is the fact that it was originally designed and

implemented on a VAX 11/780. Although GRAFCORE has been modified to operate on a VAX, it was originally designed for use on completely different machines. This means that some of the basic organization and structure of GRAFCORE is a reflection of the structure and operating characteristics of these machines. The package was also originally written in LRLTRAN, so part of the task in adapting GRAFCORE to operation on the VAX is a conversion of the software into standard FORTRAN. There is a pre-compiler for this purpose, but some of the files AFIT obtained were not pre-compiled. While this is not a fault of the system package, it illustrates some of the problems that can occur when trying to move software between different machines.

Chappell's survey did not come to my attention until I was well into my investigation, and there was not time to investigate any of the systems that had not already been uncovered. In fact, it was the article on GWCORE that led me to Chappell's survey, and I found that the four systems I had already investigated were also included in his report.

As mentioned in the introduction to this chapter, some of the systems in the survey were designed for specific graphics devices, could only be obtained at relatively high prices, or were implemented in a language not available on the VAX. Eliminating these packages, along with the four described in this chapter, leaves only five systems from Table I that have not been investigated: Hartford Graduate

Center, Los Alamos Lab, University of Alberta, University of Utah, and Western Michigan University. While the survey may now be out of date, none of these five systems listed a VAX as a host computer. On the basis of the superficial information contained in the survey forms, I would conclude that these five systems are probably no better than the GRAFCORE or GWCORE packages.

Summary

There have been quite a number of standardized graphics systems implemented since the Core System proposal was first published in 1977. This chapter has examined the systems that were most applicable to the AFIT VAX and for which current information was available. GRAFCORE, from the Lawrence Livermore Laboratory, and GWCORE, from The George Washington University, are two graphics systems that most closely met our needs. Detailed information was available on GRAFCORE when this investigation began and an effort was already underway to obtain this package. Therefore, GRAFCORE was chosen as the output-package to be used for this project.

VI Implementation of GRAFCORE on the AFIT VAX

The reasons for selecting the GRAFCORE package of output routines for installation on the AFIT VAX have been explained. This chapter describes the installation effort. The method used to obtain the GRAFCORE package is presented, followed by a brief description of the available files. Next, the steps required for the implementation are outlined in detail. Finally, a report is made of the problems that were encountered during the installation and what was done to solve them.

Obtaining GRAFCORE

As pointed out in Chapter V, plans had been made at AFIT to obtain a copy of GRAFCORE when this investigation was just beginning. This was actually one of the strongest factors in the decision to use this package. Because a version of GRAFCORE was already operating on a VAX at Lawrence Livermore, the actual installation of the package on the AFIT VAX appeared to be a minor task; the major effort would be the device driver and input-routine implementations. This turned out not to be the case. The procedure that was used to obtain the VAX version was not satisfactory and created many problems and delays in the implementation.

The GRAFCORE package was obtained through a multi-step process from library files on one of the mainframes at LLNL instead of directly from the VAX. The GRAFCORE files for the VAX version are in a library file called VAXLIB that is available on tape on one of the CDC 7600's at LLNL. The first step was to read the library tape on the 7600 and transfer selected files to a DEC-10 at LLNL. Next, the files were transfered via ARPANET to a DEC-10 at the Avionics Lab on base, where they were written back onto tape. The tape was then transferred to the Electronic Warfare Simulation Analysis Facility VAX, also at the Avionics Lab. The initial editing and installation of GRAFCORE was to be done on the Avionics Lab VAX because of its greater resources. Once the package was operational, the files would be written on floppy disks for the final transfer to the AFIT VAX.

There is really nothing wrong with this method of obtaining the software, but in this case, it resulted in a sigificant delay in the implementation effort. The first delay was caused by slow response on the part of LLNL personnel in making available an updated GRAFCORE tape for mounting on the 7600. Some of the other delays were due to one or the other of the DEC-10's being down and to problems in accessing ARPANET. As a result of these problems, the acquisition of the graphics package was delayed until 1 September. Even then, there were additional delays caused

by transmission errors and the failure to obtain all of the necessary files. Considering the problems that were encountered, a much more efficient method would have been simply to request that someone at LLNL write a tape containing the appropriate VAX files and send it to us.

The GRAFCORE Package

Table II contains a listing of the various files that are contained in the library VAXLIB. There are basically three types of files: GRAFCORE source files written in the original LRLTRAN language (generally prefixed with an 'S'), processing files for converting to standard FORTRAN and generating VAX-compatible files, and files that have already been converted and are VAX-compatible (generally prefixed with an 'X'). In addition, there are a couple of files that contain information on implementation procedures.

Due to the large size of many of the files (several thousand lines each) and the time required to transfer files over ARPANET, it was not practical to transfer the entire library. (This is another reason why having a tape shipped to us would have been more efficient.) Therefore, a selection had to be made as to which of the files to transfer. As a minimum, the files containing the BASELIB and J-level and K-level functions were required. Also, files containing implementation procedures, device driver routines, and some test routines were desired. Based on

Table II

Contents of GRAFCORE Library VAXLIB

File Names	Description
C12	Test program #12 for segmentation.
CF3D	Macro file for 3-D macros.
CFGRAF	Contains directions, common block definitions.
CFVAX	Copy of CFGRAF with parameters set for VAX.
GRAFMAC	Contains system/library parameters, functions.
JUMPTEST	Test program for jump table logic.
MAKVAX	7600 instructions for generating VAX sources.
PRECOMP	CDC 7600 precompiler for expanding macros.
SGCTEST	LRLTRAN source for test demo program.
SID	LRLTRAN source for test program.
SIN	Macros which are 'include' files for VAX.
SMINDD	Instructions for implementing a device driver.
VAXCOM	VAX command file and directions for GRAFCORE.
VAXHELP	Help file for generating GRAFCORE on the VAX.
VAXTPE	CDC 7600 program to write files for VAX.
VBASLIB	VAX BASELIB source.
VC12	VAX version of C12 test program.
VSGCTEST	VAX version of SGCTEST.
XSGCIP	FORTRAN source for input functions.
XSGCJ	FORTRAN source for J-level functions.
XSGCK	FORTRAN source for K-level functions.
XSGCTEST	FORTRAN source for SGCTEST.
XSMINDD	FORTRAN source for SMINDD.

information in the available documentation, the following files were chosen to be transfered from VAXLIB:

VBASLIB	-	source file of VAX BASELIB functions (3868 lines)
XSGCJ	-	source file of J-level functions (23218 lines)
XSGCK	-	source file of K-level functions (4270 lines)
SIN	-	file of 'include' files for VAX (63 lines)
VAXCOM	-	instructions for generating GRAFCORE on VAX (1253 lines)
VAXHELP	-	additional implementation instructions (40 lines)
SMINDD	-	source file for device driver implementation (1811 lines)
VC12	-	VAX version of C12 test (segmentation) (158 lines)
VSGCTEST	-	VAX test program (228 lines)
XSGCTEST	-	test program (153 lines)
SID	-	test program (2584 lines)

Implementation Procedure

VAXCOM contains the instructions for generating GRAFCORE to run on the VAX. The first step is to create the required directories and libraries. A list of the directories and what they are supposed to contain is provided in Table III. All of the files from the GRAFCORE package should be initially put into [GRAFCORE.TAPE]. The BASEMAC library must be created using the BASEMAC.MAR

Table III

Directories and Libraries Required for GRAFCORE

Directories	Contents
GRAFCORE.CLI	VAX FORTRAN 'include' files (cliches)
GRAFCORE.SRC	Source routines to be compiled and put into libraries
GRAFCORE.SRC3D	3-D source routines to be compiled (This is a temporary arrangement while VAX 3-D routines are being debugged.)
GRAFCORE.SV	Source and Object Libraries: GRAF3D - 3-D object library GRAFLIB - obj lib compiled w/o debug GRAFLIBD - obj lib compiled with debug S3D - 3-D source library SDD - device driver source library SGCJ - J-level source library SGCK - K-level source library SGLA - A and B level source library STEST - test demo source library
GRAFCORE.TEST	Files being compiled and linked during testing
GRAFCORE.TAPE	Original concatenated source files
GRAFCORE.UTL	*.COM utility routines from VAXCOM
BASELIB.SRC	BASELIB sources to be compiled and put into libraries
BASELIB.SV	BASELIB Source and Object Libraries: BASELIB - obj lib compiled w/o debug BASELIBD - obj lib compiled with debug BASELIB - BASELIB source library BASEMAC - macro library
BASELIB.UTL	*.COM utility routines from VAXCOM

routine. This routine is the first macro routine in the VBASLIB file. It must be manually extracted from that file and put in [BASELIB.SV] prior to creation of BASEMAC. The other libraries can be created without the need for additional routines.

VAXCOM also contains several command files that are used to set up the information in the directories. These files must be manually extracted and put into separate files that are named as indicated in the comment blocks for each one. While this is being done, the disk drive mentioned in the command procedures must be changed from 'DBB1:' to the appropriate disk on the system, which for AFIT would be 'DMA1:'. Table IV contains a list of these files. BCOMPILE.COM is put into directory [BASLIB.UTL] and the remaining command files are put into [GRAFCORE.UTL]. Associated with the two separation files, SEP3D.COM and SEPARATE.COM, are two FORTRAN programs that do the actual separation. Each of these programs must be extracted from VAXCOM and placed into [GRAFCORE.UTL], then compiled and linked. They must have the same name as the corresponding command files.

Now that the directories have been created, the libraries initialized, and the command procedures enabled, the next step is to begin putting the GRAFCORE functions into the appropriate libraries. The BASELIB functions will be done first, followed by the J- and K-level functions.

Table IV

VAX Command Procedures for Implementing GRAFCORE

Procedures	Purpose
BCOMPILE.COM	Compiles the subroutines that are in [BASELIB.SRC]. The FORTRAN routines are compiled both with and without debugging. The source and object files are put into the corresponding [BASELIB.SV] libraries.
COM3D.COM	Compiles the subroutines that are in [GRAFCORE.SRC3D]. The source and object files are put in [GRAFCORE.SV] libraries.
COMPILE.COM	Compiles the subroutines that are in [GRAFCORE.SRC]. The FORTRAN routines are compiled both with and without debugging. The source and object files are put into the corresponding [GRAFCORE.SV] libraries
EXT3D.COM	Extracts 3-D source routines from the S3D library and puts them in [GRAFCORE.TEST].
EXTRACT.COM	Extracts source routines from the [GRAFCORE.SV] libraries and puts them into [GRAFCORE.TEST].
LINKC12.COM	Links the XC12 test routine.
LINKIT.COM	Links the XSGCTEST test demo program.
MOVE.COM	Uses a file of routine names (created by EXTRACT.COM) to move tested routines from [GRAFCORE.TEST] into [GRAFCORE.SRC].
MOVE3D.COM	Uses a file of routine names (created by EXT3D.COM) to move tested routines from [GRAFCORE.TEST] into [GRAFCORE.SRC3D].
SEP3D.COM	Runs the SEP3D program to separate a batch of 3-D subroutines and puts them into [GRAFCORE.SRC3D] for compilation.
SEPARATE.COM	Runs the SEPARATE program to separate a batch of FORTRAN subroutines and puts them into [GRAFCORE.SRC] for compilation. Cliches are extracted and replaced with an 'include' statement, and the cliches are put into [GRAFCORE.CLI].

The BASELIB functions can be found in file VBASLIB, from which BASEMAC.MAR was extracted earlier. SEPARATE.COM cannot handle macro routines so it will either have to be modified, or the routines must be extracted manually. Each routine should be put into a separate file in [BASELIB.SRC] and given the proper extension, either '.FOR' or '.MAR'. BCOMPILE.COM can then be used to compile/assemble the routines and place the source and object files into the appropriate libraries in [BASELIB.SV]. Source files that compile correctly will be deleted from [BASELIB.SRC] so this directory should be empty when the operation is completed successfully.

The J-level and K-level functions are handled in a similar manner. SEPARATE.COM can be used to extract the individual FORTRAN routines from the concatenated source files. The command file will give the individual files the correct names and extensions and put them into [GRAFCORE.SRC]. It also removes the cliches (sections of code used in LRLTRAN that are analagous to macros) from the routines and replaces them with VAX FORTRAN 'include' statements. The cliches are written into [GRAFCORE.CLI]. COMPILE.COM can then be used to compile all the individual routines in [GRAFCORE.SRC] and place the source and object files into the appropriate libraries in [GRAFCORE.SV]. Just as with the BASELIB functions, source files that compile correctly will be deleted from [GRAFCORE.SRC]. Other

routines, such as those for device drivers or input functions, can be added to the system at any time by following these procedures.

There are separate directories, libraries, and command procedures for 3-D routines. These were made separate because the 3-D routines are under development and have the same names as the 2-D routines that they will replace. No 3-D routines were contained in the initial set of GRAFCORE files that were acquired, but some of the 3-D routines were included in a later acquisition. There are some bugs in these routines, one of which is that text is written backwards. According to Bryan Lawver at LLNL, a debugged set of 3-D routines should be available shortly after the first of the year. The 3-D routines are installed on the VAX by following the same procedures, but using the 3-D command procedures instead.

The only part of the basic GRAFCORE system still missing is the device driver routines. These routines need to be written according to the guidelines laid out in the file SMINDD, taking into account the capabilities of the particular graphics output device to be used. When completed, the device driver routines can be entered into the GRAFCORE libraries using the same steps as discussed in this section.

If any of the routines need to be modified at any time, the EXTRACT or EXT3D procedures can be used to remove the source routines from the libraries. The routines will be placed into [GRAFCORE.TEST] where they can be modified, compiled, linked, and tested. When the modifications have been verified, the files can be moved into [GRAFCORE.SRC] by the MOVE procedure and then compiled and put back into the libraries following the same steps as performed initially.

Problems Encountered

The long delay in acquiring the GRAFCORE package was only the first of many problems encountered during this project. The directories and libraries were created without any problem. The first difficulty arose when using BCOMPILE.COM to compile the BASELIB routines for entry into the BASELIB libraries. File-name errors were detected during the calls to the FORTRAN compiler and MACRO assembler. The errors were eventually traced to the fact that the BCOMPILE.COM file was entirely in lower case. Normally, this would not be a problem since commands can be entered in either upper or lower case, with the lower case being converted to upper case. However, when this procedure searched the file names in [BASELIB.SRC] looking for their extensions, it used the search strings ".for" and ".mar". In this case, the strings were taken literally and not converted to upper case. Since the file names are stored in

upper case, these strings were never found. As a result, the extensions were included as part of the file name. Thus, when the compiler or assembler was called and a '.for', '.mar', or '.obj' extension was concatenated onto the file name, an error was detected because the names already contained extensions. By changing the literal search strings for the extensions to upper case, this problem was corrected. In addition, another search string, " no files found.", had to be changed to " No Files Found.". The search strings in all the other command procedures in VAXCOM, which were also all lower case, were examined for correctness and changed if necessary.

The lower case problem was caused by one of the routines on the 7600. Somewhere in the creation process VAXCOM existed as a file of 6-bit ASCII characters. A 7600 routine converts the 6-bit ASCII to 8-bit ASCII and for some reason, it defaults to lower case characters.

After correcting BCOMPILE.COM, the FORTRAN routines compiled correctly, but many of the macro routines would not. The problem here turned out to be transmission errors. The circumflex character ('^'), used to indicate unary operators, had been lost in transmission and was missing from all the routines. A second problem was that some of the logical inclusive OR operators ('!') had been converted into carriage returns during transmission. This was caused by either the CDC 7600 or by the DEC-10's. One of these

machines treats an exclamation point in a certain column as a carriage return so this conversion was made during the transmission. After correcting these errors, the routines were assembled and placed into the libraries without error.

Placing the J-level and K-level routines into the [GRAFCORE.SV] libraries was the next step in the implementation. SEPARATE.COM was to be used here, but it was discovered that the cliches in the J and K functions had already been removed and replaced with 'include' statements. The XSGCJ and XSGCK files were the only J and K files in the VAXLIB library, so it was assumed that they had to be the correct files for these routines. On the other hand, it does not make sense for a command procedure to be written to remove cliches when they were, in fact, already removed. This was an indication that maybe those two routine files were not the correct ones after all. In any case, this was not a major problem. The XSGCJ and XSGCK files are the files that would result after running SEPARATE.COM, so there would be no advantage in that regard to obtaining different files. SEPARATE.COM was easily modified to ignore the cliches and still separate the concatenated source files as it was originally intended to do.

Of greater concern was the fact that the cliches that should have been removed and placed into [GRAFCORE.CLI] were not contained in any of the GRAFCORE files that had been transferred. The file SIN had been described as containing

macros that were the 'include' files for the VAX, but it contained nothing more than a series of 'USE' statements, which are LRLTRAN statements.

A call to Jeff Rowe, of the Computer Graphics Group at LLNL, cleared up our problem. He stated that the cliches were contained in files CFVAX and GRAFMAC and that there were two methods that could be used to obtain them. First, the file SIN could be precompiled on the CDC 7600, which would use CFVAX and GRAFMAC to create the proper cliche ('include') files in SIN. Then the precompiled SIN file could be transferred via ARPANET as had been done with the rest of the package. The second method was to ignore SIN altogether and transfer CFVAX and GRAFMAC, then maually extract the 'include' files using a text editor. The latter method was chosen and the two new files were finally received in mid-October.

CFVAX and GRAFMAC are both long files and they each contain much GRAFCORE documentaion on common storage and variable descriptions. They are basically a collection of cliches, which are LRLTRAN versions of macros. The blocks of statements which make up the code for the 'include' statements are easily identified, as they are delimited by 'C BEGINCLICHE (name)' and 'C END CLICHE'. CFGRAF contains 57 cliches and GRAFMAC contains 135, but altogether only eight of the cliches contain the 'BEGINCLICHE-END CLICHE' blocks.

Just removing these eight blocks of statements and putting them into [GRAFCORE.CLI] is not enough. CFVAX and GRAFMAC are written in LRLTRAN, and these 'include' files must be further processed to convert them into standard FORTRAN. CFVAX and GRAFMAC were designed to be run on the CDC 7600 during precompilation of LRLTRAN source files to create VAX-compatible files. CFVAX is a copy of CFGRAF with parameters set for the VAX. CFGRAF is a file that contains 18 parameters used for creating files for various machines. Files can be created for the 7600, STAR, CRAY, VAX, and LSI-11; in FORTRAN or LRLTRAN; with or without comments; using six- or eight-character BASELIB names; and so on for several other parameters.

The parameters control the contents of the precompiled files through the use of do-not-compile-if (DIF) statements. These statements are of the form:

```
DIF(parameter)A,B,C
```

This is similar to an arithmetic IF statement in FORTRAN. 'A' is used if the parameter is negative, 'B' is used if the parameter is zero, and 'C' is used if the parameter is positive. 'A', 'B', and 'C' can be either integers or labels. If integers, it means not to compile that many statements following the DIF, and if labels, it means not to compile the statements between the DIF and the label.

Manually searching the files and deleting statements according to the DIF instructions is a simple, but time-consuming, task. When this is done, it should be done for the entire cliche within which the 'BEGINCLICHE-END CLICHE' blocks are located. Some of the cliches will contain statements that affect these blocks. For example, in the following section of code

```
          CLICHE CFLINK (CFGRAF)
                      .
                      .
          PARAMETER      (MAXPDF=7)
          PARAMETER      (MAXPD=10)
                      .
                      .
      C BEGINCLICHE CFLINK
                      .
                      .
          COMMON /GCDJMP/  KEY25, LKEY25, LOCBAS(1),
         1    JPDTAB(MAXPD,MAXPDF)
                      .
                      .
      C END CLICHE
                      .
                      .
```

the values of the parameters that appear in the main body of the cliche have to be substituted for the parameter names that appear in the common statement within the block of statements to be extracted.

Another task that needed to be done was to replace calls to a LRLTRAN macro with the equivalent FORTRAN statements. The macros involved were VECTORR and a macro that it calls, VEKTOR. These two macros (cliches) can be found in the file GRAFMAC. The purpose of these macros is

to create a length-address pair of words for a given descriptor name and place them into a common area. The call to VECTORR has the form:

VECTORR [common] [name]

The equivalent FORTRAN code is:

```
INTEGER  Dname, Aname, Lname
DIMENSION  Dname(2)
COMMON /common/  Dname
EQUIVALENCE  (Dname(1),Lname), (Dname(2),Aname)
```

Once the 'include' statement blocks had been properly modified, they were placed into [GRAFCORE.CLI]. SEPARATE.COM was modified so it would only extract the individual routines from XSGCJ and XSGCK and ignore handling the cliches. After the individual J and K routines were separated and put into [GRAFCORE.SRC], COMPILE.COM was run to compile the routines and put them into the libraries. All of the routines compiled successfully except for KBDAT4 and KIDTDV.

KBDAT4 contained some errors in a data statement as if there may be some variables missing. A call to Bryan Lawver cleared up this error. It turned out that there was an extra DATA statement that did not belong in the routine.

Bryan said that when GRAFCORE was installed on the VAX, the original routines were brought to the VAX and edited on that system and then returned to the 7600 library. He stated that similar problems due to errors made during editing should be expected, although no others have been found.

The problem with KIDTDV had been anticipated, but has turned out to be more serious than first suspected. The error here was that the file contained a LRLTRAN USE statement and, in addition, did not contain an 'END' statement. The missing 'END' statement had been noticed shortly after the package was first received in September, and this had led to doubts about the completeness of the XSGCK file, since KIDTDV was the last routine in the file. Thinking that part of the file may have been lost in transmission, the file was checked on the 7600 during the process of obtaining copies of CFVAX and GRAFMAC. The XSGCK file was the same on the 7600 as the copy we had. Unfortunately, the completeness of the 7600 XSGCK file was not verified with LLNL personnel at the time. Not until after the problem with the cliches and 'include' files was solved at the end of October was a call made to Bryan Lawver at LLNL. He confirmed that over half of the XSGCK file was missing, and agreed to send a tape with a complete copy of the K-level routines. This tape is also to include some 3-D functions and a Tektronix device driver.

Summary

Since GRAFCORE is operating on a VAX at Lawrence Livermore it should have been easy to transfer to the AFIT VAX. The process of acquiring GRAFCORE from the tape libraries on the CDC 7600, instead of directly from the LLNL VAX, left a lot to be desired. Transferring the software from machine to machine caused a lengthy delay in receiving the graphics package and transmission errors affected some of the files. The file containing the K-level routines was incomplete and some necessary files, CFVAX and GRAFMAC were not initially transferred. A number of useful utilities have been provided for performing many of the needed file operations, but the failure to insure that all the files had been precompiled before transmission resulted in a lot of man-hours spent manually processing the software.

VII Conclusions and Recommendations

The preceeding six chapters have covered the efforts undertaken in this project in working towards the implementation of a standard graphics package for the VAX in the Digital Engineering Lab at AFIT. The initial goals for the project were presented in Chapter I. Chapter II presented a description of basic graphics principles. Chapter III presented general graphics requirements and the more detailed requirements of the Core System proposal for standardized graphics were discussed in Chapter IV. Chapter V examined currently available standard graphics packages and presented the reasons for selecting the GRAFCORE package of routines for implementation at AFIT. The procedures for implementing GRAFCORE and the problems encountered in doing so were detailed in Chapter VI. This chapter discusses the overall results of this project, provides a summary of the current status of the effort, and presents recommendations for continued work in this area.

Conclusions

This project began as a continuation of work begun by Capt Curling on a design of a set of standardized input routines. His design had been completed, but before it could be implemented, a set of standardized output routines would have to be installed on the VAX. Capt Curling had

proposed using an existing output-routine package to interface with his input routines. Acquisition and implementation of an output package was the first goal set for this project.

Installing an exsiting graphics-output package on the VAX was thought to be a minor effort, especially since it was known that at least one package (GRAFCORE) was running on another VAX. Therefore, other goals were set for this project that were to be the main thrust of the investigation. The first was to provide the graphics package with device drivers for the Tektronix 4014 terminal and the ISC 8001G color raster terminal. Then would come the implementation of Capt Curling's input routines. Part of this implementation would be an expansion to accommodate the ISC terminal since Curling's design had been tailored specifically for the Tektronix terminal.

None of these latter goals was met because of the unforseen problems encounterd in the implementation of the graphics-output package, which turned that task into a major effort. The lengthy delay in acquiring the software was not, by itself, the biggest problem. But combined with the missing files, transmission errors, and the need for manual processing of the files resulting from the manner in which GRAFCORE was acquired, the total delay proved to be significant.

Part of the problem was the unfamiliarity with the large GRAFCORE/GRAFLIB system and the lack of knowledge about which specific files were needed. By maintaining closer contact with the personnel at LLNL, many of the problems could most likely have been minimized or eliminated, and much time and effort would have been saved.

One advantage that came out of the acquisition process that was used is the fact that the files CFVAX and GRAFMAC were obtained. These two files contain much detailed documentation on GRAFCORE that might otherwise not have been available. Whether these files would have been included in a GRAFCORE tape written by LLNL personnel for AFIT's use is not known. These files are only needed to run the precompiler on the CDC 7600 to generate the VAX files and they do not run on the VAX itself. In addition, the examination of these two files and the requirement to perform much manual processing of the software has led to a better understanding of the structure and operation of GRAFCORE as well as the LRLTRAN language and the precompiling process.

A lot of effort has gone into the design of GRAFCORE and the development is still continuing. The examples in the documentation indicate that GRAFCORE, along with the higher levels of GRAFLIB, forms a very powerful package. However, the fact cannot be ignored that this graphics system was originally designed and implemented for host

machines other than the VAX. All of the documentation is aimed at users of the CRAY, STAR, and 7600 machines and the LRLTRAN language. There is even a note in the file VAXCOM that explains that the VAX version of BASELIB contains many simulations "Because BASELIB was originally designed to run on an LTSS operating system . . ." and "Due to system incompatibilities between LTSS and VMS, the VAX version of BASELIB is actually only an approximation of the LTSS versions".

When GRAFCORE and GWCORE were discussed in Chapter V, a statement was made to the effect that if this project could be started over, with full knowledge of the problems that would be ecountered, the GWCORE package would have been selected instead. GWCORE was designed and implemented on the VAX, so all documentation pertains to the VAX and there are no simulations or approximations needed in the software because of differences in operating systems. The George Washington package has a definite advantage in this area over the Lawrence Livermore package. GWCORE is a more complete, but stricter, implementation of the Core System proposal. Whether or not stricter adherance to the proposal is an advantage would have to be decided through actual use of the two packages. The results would probably be quite subjective.

Recommendations

All the J-level and K-level routines that were first obtained have been compiled and entered into the libraries and the remaining K-level routines were added as soon as the new tape arrived from LLNL. The other GRAFCORE files have been corrected where necessary so the output package should be in working order.

The next task should be to begin work on device drivers to interface with the output package. Guidelines can be found in the file SMINDD. Following this, the implementation of the input routines can begin.

The GRAFCORE package as currently installed on the VAX provides 2-D and some 3-D output functions, although as mentioned in Chapter VI, there are still some bugs in the 3-D routines. Contact should be maintained with personnel of the Computer Graphics Group at LLNL so that the routines for higher level functions, both input and output, may be obtained as they become available. An updated set of 3-D routines is to become available after the first of the year. Input routines may not be available for some time since the LLNL VAX currently has no requirement for input functions. Getting the Level A and Level B sections of GRAFLIB would also be a possibility. Names and phone numbers for LLNL personnel can be found in the file VAXCOM.

As an alternative, rather than use the additional LLNL functions, future efforts could be undertaken to design and implement these additional functions at AFIT. A start could be made in this direction by interfacing Capt Curling's input design with the GRAFCORE output package, since LLNL's implementation of the input routines is not expected to proceed quickly.

Discussions in this report on GRAFCORE and GWCORE have been concerned with whether one or the other should have been selected for implementation on the VAX. Actually, there is no reason why both of them should not be obtained. Bill McQuay, AFWAL/AAWA/54429, who supervises the VAX at the Electronic Warfare Simulation Analysis Facility, has received the GWCORE package and a copy should be obtained for AFIT. This is a February 1981 version and does not contain any input functions. Input routines are being developed by GW, and they can be obtained through Bill McQuay when they become available, which should be very soon. Instead, as was suggested for GRAFCORE, a project could be undertaken at AFIT to implement a set of input routines for GWCORE based on the design of Capt Curling.

The GWCORE package could be used mainly for application program development, while GRAFCORE could provide a basis for development of additional basic functions, as mentioned previously. Device driver development would be required for both systems.

When GRAFCORE has been expanded to a level more equivalent to that of GWCORE a better comparison can be made between the two. The fact that GWCORE was designed for the VAX, and GRAFCORE was not, could be investigated to see if that is a significant advantage. Also, the fact that the two packages provide different implementations of the Core Standard could form the basis of an investigation into the good and bad points of that standardized proposal.

Bibliography

1. Bergeron, Daniel, _et al_. "Graphics Programming Using the Core System," _Computing Surveys_, _10_: 389-443 (December 1978).

2. Chappell, Gary. "Implementations of the CORE," _Computer Graphics_, _13_ (4): 260-278 (February 1980).

3. Curling, Harold L. _Design of an Interactive Input Graphics System Based on the ACM Core Standard._ MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1980. (AFIT/GCS/EE/80D-6).

4. Finch, Walter L. Product Manager, National Technical Information Service (personal correspondence). Springfield, Virginia, June 29, 1981.

5. Foley, James D. "A Standard Computer Graphics Subroutine Package," _Computers & Structures_, _10_: 141-147 (April 1979).

6. Foley, James D. and Patricia A. Wenner. "The George Washington University Core System Implementation," _Computer Graphics_, _15_ (3): 123-131 (August 1981).

7. Herman, Gene. "The Core System of Graphics is Complete -- for now," _INPUT/OUTPUT: Grinnell College Computer Services Newsletter_, _5_ (4): 1-3 (April 10, 1981).

8. Keller, Pete, _et al_. _GRAFLIB Reference Manual_. LCSD-432. Livermore, California: Computer Graphics Group, Lawrence Livermore Laboratory, October 24, 1980.

9. Michener, James C. and Andries Van Dam. "A Functional Overview of the Core System with Glossary," _Computing Surveys_, _10_: 381-387 (December 1978).

10. Michener, James C. and James D. Foley. "Some Major Issues in the Design of the Core Graphics System," _Computing Surveys_, _10_: 445-463 (December 1978).

11. Newman, William M. and Andries Van Dam. "Recent Efforts Towards Graphics Standardization," _Computing Surveys_, _10_: 365-380 (December 1978).

12. Newman, William M. and Robert F. Sproull. _Principles of Interactive Computer Graphics_ (Second Edition). New York: McGraw-Hill Book Compay, 1979.

13. O'Hair, Kelly, _et al_. _Computer Graphics By Example: GRAFLIB_ . LCSD-423. Livermore, California: Computer Graphics Group, Lawrence Livermore Laboratory, October 24, 1980.

14. -----. _Computer Graphics By Example: GRAFCORE_. LCSD-433. Livermore, California: Computer Graphics Group, Lawrence Livermore Laboratory, November 14, 1980.

15. -----. _GRAFLIB Beginner User Manual_. LCSD-424. Livermore, California: Computer Graphics Group, Lawrence Livermore Laboratory, January 16, 1981.

16. -----. _GRAFLIB: Graphics By Example (Level B)_. LCSD-426. Livermore, California: Computer Graphics Group, Lawrence Livermore Laboratory, January 26, 1981.

17. -----. _GRAFLIB Advanced User Manual_. LCSD-425. Livermore, California: Computer Graphics Group, Lawrence Livermore Laboratory, March 14, 1981.

18. Rowe, Jeffery, _et al_. _GRAFCORE Reference Manual_. LCSD-434. Livermore, California: Computer Graphics Group, Lawrence Livermore Laboratory, November 15, 1980.

19. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH," _Computer Graphics_, _11_ (3): (Fall 1977).

20. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH," Computer Graphics, 13 (3): (August 1979).

21. Wenner, Patricia A., et al. Design Document for the George Washington University Implementation of the 1979 GSPC CORE System. Technical Report GWU-EE/CS-80-06. Washington, D.C.: Department of Electrical Engineering and Computer Science, The George Washington University, May 1980.

Appendix A

GRAFCORE J-level Functions

The J-level functions are the actual GRAFCORE implementation of the Core System proposal. This appendix contains a listing of the J-functions as well as the Core proposal functions. The listing is divided into three sections: J-functions for which there are corresponding Core functions, J-functions for which there are no Core functions, and Core functions for which there are no J-functions. Functions which have been implemented but are not included in our version of the GRAFCORE package are marked with an asterisk ('*'). J-level functions that have been named but not yet implemented are marked (NI).

Contents

Contents

Contents

I Core Functions and The Corresponding J-Functions

This section contains a listing of the Core System functions for which J-level functions have been designated. In most cases, the only difference between the J-functions and the Core functions is in the function and parameter names. Some of the J-functions do, however, have a different parameter count than the corresponding Core functions. Also, there are some differences in the number of functions used to implement a specific task. For example, the Core Standard might have separate functions for 2-D and 3-D but there will be only one J-function that can be used for either 2-D or 3-D (see 4.2.1.1). Another example would be where a single Core function has been implemented as two or three separate J-functions (see 4.2.2.3). The numbering used is the same as that used in the Core System proposal [Ref 20:II-20 - II-103,III-13]. Each Core function is listed first, followed by the corresponding J-function(s).

2.0 Output Primitives

2.2.1 Current Position

2.2.1.1 Move

```
MOVE_ABS_2 (X,Y)
JSCP2A (XWC,YWC)

MOVE_ABS_3 (X,Y,Z)
JSCP3A (XWC,YWC,ZWC)
```

```
    MOVE_REL_2 (DX,DY)
(NI)JSCP2R

    MOVE_REL_3 (DX,DY,DZ)
(NI)JSCP3R
```

2.2.1.2 Inquire Current Position

```
    INQUIRE_CURRENT_POSITION_2 (X,Y)
    JICP2D (X,Y)

    INQUIRE_CURRENT_POSITION_3 (X,Y,Z)
(NI)JICP3D
```

2.2.2 <u>Line-Drawing Primitives</u>

2.2.2.1 Line

```
    LINE_ABS_2 (X,Y)
(NI)JPLN2A

    LINE_ABS_3 (X,Y,Z)
    JPLN3A (X1,Y1,Z1,X2,Y2,Z2)

    LINE_REL_2 (DX,DY)
(NI)JPLN2R

    LINE_REL_3 (DX,DY,DZ)
(NI)JPLN3R
```

2.2.2.2 Polyline

```
    POLYLINE_ABS_2 (X_ARRAY,Y_ARRAY,N)
    JPPL2A (XARRAY,YARRAY,N)

    POLYLINE_ABS_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)
    JPPL3A (X,Y,Z,N)

    POLYLINE_REL_2 (DX_ARRAY,DY_ARRAY,N)
(NI)JPPL2R

    POLYLINE_REL_3 (DX_ARRAY,DY_ARRAY,DZ_ARRAY,N)
(NI)JPPL3R
```

2.2.3 <u>Text Primitives</u>

2.2.3.1 TEXT (CHARACTER_STRING)
JPTX2D (ITEXT,LTEXT,ICHAR)

2.2.3.2 Inquire Text Extent

```
    INQUIRE_TEXT_EXTENT_2 (CHARACTER_STRING,
                           SURFACE_NAME,DX,DY)
(NI)JITX2D

    INQUIRE_TEXT_EXTENT_3 (CHARACTER_STRING,
                        SURFACE_NAME,DX,DY,DZ)
(NI)JITX3D
```

2.2.4 Marker Primitives

2.2.4.1 Marker

```
    MARKER_ABS_2 (X,Y)
(NI)JPMK2A

    MARKER_ABS_3 (X,Y,Z)
(NI)JPMK3A

    MARKER_REL_2 (DX,DY)
(NI)JPMK2R

    MARKER_REL_3 (DX,DY,DZ)
(NI)JPMK3R
```

2.2.4.2 Polymarker

```
    POLYMARKER_ABS_2 (X_ARRAY,Y_ARRAY,N)
    JPPM2A (XARRAY,YARRAY,N)

    POLYMARKER_ABS_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)
(NI)JPPM3A

    POLYMARKER_REL_2 (DX_ARRAY,DY_ARRAY,N)
(NI)JPPM2R

    POLYMARKER_REL_3 (DX_ARRAY,DY_ARRAY,DZ_ARRAY,N)
(NI)JPPM3R
```

2.2.5 Raster Extensions - Polygon

```
    POLYGON_ABS_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)
    JPPG3A (XARRAY,YARRAY,ZARRAY,N)
```

3.0 <u>Picture Segmentation and Naming</u>

3.2.1 <u>Retained Segments</u>

3.2.1.1 CREATE_RETAINED_SEGMENT (SEGMENT_NAME)
JNSGDA (ISEGNAM)

3.2.1.2 CLOSE_RETAINED_SEGMENT ()
JESGDA (ISEGNAM)

3.2.1.3 DELETE_RETAINED_SEGMENT (SEGMENT_NAME)
JESGXX (ISEGNAM)

3.2.1.4 DELETE_ALL_RETAINED_SEGMENTS ()
(NI)JESGGL

3.2.1.5 RENAME_RETAINED_SEGMENT (SEGMENT_NAME,NEW_NAME)
JSSGNM (NAMSEG,NEWSEG)

3.2.1.6 INQUIRE_RETAINED_SEGMENT_SURFACES (SEGMENT_NAME,
ARRAY_SIZE,VIEW_SURFACE_ARRAY,NUMBER_OF_SURFACES)
(NI)JISGVS

3.2.1.7 INQUIRE_RETAINED_SEGMENT_NAMES (ARRAY_SIZE,
SEGMENT_NAME_ARRAY,NUMBER_OF_SEGMENTS)
(NI)JISGGL

3.2.1.8 INQUIRE_OPEN_RETAINED_SEGMENT (SEGMENT_NAME)
(NI)JISGXX

3.2.2 <u>Temporary Segments</u>

Note: The same J-functions are used for both retained and temporary segments. The segment type is specified separately by JSSGTP.

3.2.2.1 CREATE_TEMPORARY_SEGMENT ()
JNSGDA (ISEGNAM)

3.2.2.2 CLOSE_TEMPORARY_SEGMENT ()
JESGDA (ISEGNAM)

3.2.2.3 INQUIRE_OPEN_TEMPORARY_SEGMENT (OPEN)
(NI)JISGXX

4.0 Attributes

4.2.1 Static Attributes

4.2.1.1 Setting Primitive Static Attribute Values

```
         SET_COLOR (COLOR)
         JSCRXX (ICOLOR)

         SET_INTENSITY (INTENSITY)
     (NI)JSATIN

         SET_LINESTYLE (LINESTYLE)
         JSLNFT (LISTYLE,SCALE)

         SET_LINEWIDTH (LINEWIDTH)
         JSLNSZ (WIDTH)

         SET_PEN (PEN)
     (NI)JSPNXX

         SET_FONT (FONT)
         JSTXFT (IFONT)

         SET_CHARSIZE (CHARWIDTH,CHARHEIGHT)
         JSTXSZ (CHARWIDTH,CHARHEIGHT)

         SET_CHARPLANE (DX_PLANE,DY_PLANE,DZ_PLANE)
         JSTXPE (DXPLAN,DYPLAN,DZPLAN)

         SET_CHARUP_2 (DX_CHARUP,DY_CHARUP)
         SET_CHARUP_3 (DX_CHARUP,DY_CHARUP,DZ_CHARUP)
         JSTXUP (DXUP,DYUP,DZUP)

         SET_CHARPATH (CHARPATH)
         JSTXLN (IPATH)

         SET_CHARSPACE (CHARSPACE)
         JSTXSP (CSPACE)

         SET_CHARJUST (CHARJUST)
         JSTXJF (JFVERT,JFHORZ)

         SET_CHARPRECISION (CHARPRECISION)
         JSTXLW (IQUAL)
         JSTXHI (IQUAL)

         SET_MARKER_SYMBOL (SYMBOL)
         JSMKXX (MARKER)

         SET_PICK_ID (ID)
     (NI)JSOPID
```

4.2.1.2 SET_PRIMITIVE_ATTRIBUTES_2 (PRIMITIVE_ATTRIBUTE_ARRAY_2)
SET_PRIMITIVE_ATTRIBUTES_3 (PRIMITIVE_ATTRIBUTE_ARRAY_3)
(NI)JSPRAT

4.2.1.3 Inquiring Primitive Static Attribute Values

```
    INQUIRE_COLOR (COLOR)
    JICRXX (ICOLOR)

    INQUIRE_INTENSITY (INTENSITY)
(NI)JIATIN

    INQUIRE_LINESTYLE (LINESTYLE)
    JILNFT (LISTYLE,SCALE)

    INQUIRE_LINEWIDTH (LINEWIDTH)
    JILNSZ (WIDTH)

    INQUIRE_PEN (PEN)
(NI)JIPNXX

    INQUIRE_FONT (FONT)
(NI)JITXFT

    INQUIRE_CHARSIZE (CHARWIDTH,CHARHEIGHT)
    JITXSZ (CHARWIDTH,CHARHEIGHT)

    INQUIRE_CHARPLANE (DX_PLANE,DY_PLANE,DZ_PLANE)
    JITXPE (DXPLAN,DYPLAN,DZPLAN)

    INQUIRE_CHARUP_2 (DX_CHARUP,DY_CHARUP)
    INQUIRE_CHARUP_3 (DX_CHARUP,DY_CHARUP,DZ_CHARUP)
    JITXUP (DXUP,DYUP,DZUP)

    INQUIRE_CHARPATH (CHARPATH)
    JITXLN (IPATH)

    INQUIRE_CHARSPACE (CHARSPACE)
    JITXSP (CHARSPAC)

    INQUIRE_CHARJUST (CHARJUST)
    JITXJF (JFVERT,JFHORZ)

    INQUIRE_CHARPRECISION (CHARPRECISION)
(NI)JITXPR

    INQUIRE_MARKER_SYMBOL (SYMBOL)
    JIMKXX (MARKER)

    INQUIRE_PICK_ID (ID)
(NI)JIOPID
```

4.2.1.5 SET_IMAGE_TRANSFORMATION_TYPE (TYPE)
JSSGTP (ISEGTP)

4.2.1.6 INQUIRE_IMAGE_TRANSFORMATION_TYPE (TYPE)
(NI)JISGTP

4.2.1.7 INQUIRE_SEGMENT_IMAGE_TRANSFORMATION_TYPE (SEGMENT_NAME,TYPE)
(NI)JISGZZ

4.2.2 Retained Segment Dynamic Attributes

4.2.2.3 Setting A Retained Segment's Dynamic Attribute Values

SET_SEGMENT_VISIBILITY (SEGMENT_NAME,VISIBILITY)
(NI)JSSGVI

SET_SEGMENT_HIGHLIGHTING (SEGMENT_NAME, HIGHLIGHTING)
(NI)JSSGHL

SET_SEGMENT_DETECTABILITY (SEGMENT_NAME, DETECTABILITY)
(NI)JSSGDY

SET_SEGMENT_IMAGE_TRANSLATE_2(SEGMENT_NAME,TX,TY)
JSIT2T (NAMSEG,XTRAN,YTRAN)

SET_SEGMENT_IMAGE_TRANSFORMATION_2 (SEGMENT_NAME, SX,SY,A,TX,TY)
JSIT2S (NAMSEG,SCALEX,SCALEY)
JSIT2R (NAMSEG,ANGLE)

SET_SEGMENT_IMAGE_TRANSFORMATION_3 (SEGMENT_NAME, SX,SY,SZ,AX,AY,AZ,TX,TY,TZ)
(NI)JSIT3S
(NI)JSIT3R
(NI)JSIT3T

4.2.2.4 Inquiring A Retained Segment's Dynamic Attribute Values

INQUIRE_SEGMENT_VISIBILITY (SEGMENT_NAME, VISIBILITY)
(NI)JISGVI

INQUIRE_SEGMENT_HIGHLIGHTING (SEGMENT_NAME, HIGHLIGHTING)
(NI)JISGHL

```
    INQUIRE_SEGMENT_DETECTABILITY (SEGMENT_NAME,
                                        DETECTABILITY)
(NI)JISGDY

    INQUIRE_SEGMENT_IMAGE_TRANSLATE_2 (SEGMENT_NAME,
                                                TX,TY)
    JIIT2T (NAMSEG,XTRAN,YTRAN)

    INQUIRE_SEGMENT_IMAGE_TRANSFORMATION_2
                          (SEGMENT_NAME,SX,SY,A,TX,TY)
    JIIT2S (NAMSEG,SCALEX,SCALEY)
    JIIT2R (NAMSEG,ANGLE)

    INQUIRE_SEGMENT_IMAGE_TRANSFORMATION_3
            (SEGMENT_NAME,SX,SY,SZ,AX,AY,AZ,TX,TY,TZ)
(NI)JIIT3S
(NI)JIIT3R
(NI)JIIT3T
```

5.0 <u>Viewing Operations and Coordinate Transformations</u>

5.2.1 <u>2-D Viewing Operation</u>

```
5.2.1.1  SETWINDOW (XMIN,XMAX,YMIN,YMAX)
         JSWD2D (XMIN,XMAX,YMIN,YMAX)

5.2.1.2  SET_VIEW_UP_2 (DX_UP,DY_UP)
     (NI)JSUP2D

5.2.1.3  SET_NDC_SPACE_2 (WIDTH,HEIGHT)
     (NI)JSNR2D

5.2.1.4  SET_VIEWPORT_2 (XMIN,XMAX,YMIN,YMAX)
         JSVP2D (XMIN,XMAX,YMIN,YMAX)
```

5.2.1.5 Inquiry For Individual Viewing Operation Parameters

```
    INQUIRE_WINDOW (XMIN,XMAX,YMIN,YMAX)
(NI)JIWD2D

    INQUIRE_VIEW_UP_2 (DX_UP,DY_UP)
(NI)JIUP2D

    INQUIRE_NDC_SPACE_2 (WIDTH,HEIGHT)
(NI)JINR2D

    INQUIRE_VIEWPORT_2 (XMIN,XMAX,YMIN,YMAX)
(NI)JIVP2D

5.2.1.6  MAP_NDC_TO_WORLD_2 (NDC_X,NDC_Y,X,Y)
     (NI)JXWD2D
```

5.2.1.7 MAP_WORLD_TO_NDC_2 (X,Y,NDC_X,NDC_Y)
(NI)JCVP2D

5.2.2 <u>3-D Viewing Operation</u>

5.2.2.1 SET_VIEW_REFERENCE_POINT (X_REF,Y_REF,Z_REF)
JSVUPT (XREF,YREF,ZREF)

5.2.2.2 SET_VIEW_PLANE_NORMAL (DX_NORM,DY_NORM,DZ_NORM)
JSVUNR (XDELTA,YDELTA,ZDELTA)

5.2.2.3 SET_VIEW_PLANE_DISTANCE (VIEW_DISTANCE)
JSVUDX (VIEWDIS)

5.2.2.4 SET_VIEW_DEPTH (FRONT_DISTANCE,BACK_DISTANCE)
JSVUDP (FRONT,BACK)

5.2.2.5 SET_PROJECTION (PROJECTION_TYPE,DX_PROJ,DY_PROJ,
DZ_PROJ)
JSPSXX (IPTYPE,XDELTA,YDELTA,ZDELTA)

5.2.2.6 SET_WINDOW (UMIN,UMAX,VMIN,VMAX)
JSWD3D (UMIN,UMAX,VMIN,VMAX)

5.2.2.7 SET_VIEW_UP_3 (DX_UP,DY_UP,DZ_UP)
JSUP3D (DXUP,DYUP,DZUP)

5.2.2.8 SET_NDC_SPACE_3 (WIDTH,HEIGHT,DEPTH)
(NI)JSNR3D

5.2.2.9 SET_VIEWPORT_3 (XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX)
JSVP3D (XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX)

5.2.2.10 SET_VIEWING_PARAMETERS (VIEWING_PARAMETER_ARRAY)
(NI)JSVTPR

5.2.2.11 Inquiry For Individual Viewing Operation Parameters

INQUIRE_VIEW_REFERENCE_POINT (X_REF,Y_REF,Z_REF)
(NI)JIVUPT

INQUIRE_VIEW_PLANE_NORMAL (DX_NORM,DY_NORM,
DZ_NORM)
(NI)JIVUNR

INQUIRE_VIEW_PLANE_DISTANCE (VIEW_DISTANCE)
(NI)JIVUDX

INQUIRE_VIEW_DEPTH (FRONT_DISTANCE,BACK_DISTANCE)
(NI)JIVUDP

```
        INQUIRE_PROJECTION (PROJECTION_TYPE,DX_PROJ,
                                             DY_PROJ,DZ_PROJ)
    (NI)JIPSXX

        INQUIRE_VIEW_UP_3 (DX_UP,DY_UP,DZUP)
    (NI)JIUP3D

        INQUIRE_NDC_SPACE_3 (WIDTH,HEIGHT,DEPTH)
    (NI)JINR3D

        INQUIRE_VIEWPORT_3(XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX)
    (NI)JIVP3D
```

5.2.2.12 INQUIRE_VIEWING_PARAMETERS(VIEWING_PARAMETER_ARRAY)
(NI)JIVTZZ

5.2.2.13 MAP_NDC_TO_WORLD_3 (NDC_X,NDC_Y,NDC_Z,X,Y,Z)
(NI)JXWD3D

5.2.2.14 MAP_WORLD_TO_NDC_3 (X,Y,Z,NDC_X,NDC_Y,NDC_Z)
(NI)JXVP3D

5.2.3 Viewing Control

5.2.3.1 SET_WINDOW_CLIPPING (ON_OFF)
JSVTCL (ICLIP)

5.2.3.2 Depth Clipping

```
SET_FRONT_PLANE_CLIPPING (ON_OFF)
SET_BACK_PLANE_CLIPPING (ON_OFF)
JSCLPE (IONOFF)
```

5.2.3.3 SET_COORDINATE_SYSTEM_TYPE (TYPE)
JSGLSP (ICTYPE)

5.2.3.4 INQUIRE_VIEWING_CONTROL_PARAMETERS(WINDOW_CLIPPING,
FRONT_CLIPPING,BACK_CLIPPING,TYPE)
JIVTPR (IWCLIP,IFCLIP,IBCLIP,ITYPE)

5.2.4 World Coordinate Transformations

5.2.4.1 SET_WORLD_COORDINATE_MATRIX_2 (MATRIX_2)
(NI)JSWDM2

SET_WORLD_COORDINATE_MATRIX_3 (MATRIX_3)
(NI)JSWDM3

5.2.4.2 INQUIRE_WORLD_COORDINATE_MATRIX_2 (MATRIX_2)
(NI)JIWDM2

INQUIRE_WORLD_COORDINATE_MATRIX_3 (MATRIX_3)
(NI)JIWDM3

6.0 Input Primitives

6.2.4 Initializing and Enabling Input Devices

6.2.4.3 ENABLE_DEVICE (DEVICE_CLASS,DEVICE_NUM)
(NI)JCDVXX

6.2.4.4 ENABLE_GROUP (DEVICE_CLASS,DEVICE_NUM_ARRAY,N)
(NI)JCCSXX

6.2.4.5 DISABLE_DEVICE (DEVICE_CLASS,DEVICE_NUM)
(NI)JDDVGL

6.2.4.6 DISABLE_GROUP (DEVICE_CLASS,DEVICE_NUM_ARRAY,N)
(NI)JDCSXX

6.2.4.7 DISABLE_ALL ()
(NI)JDEVGL

6.2.5 Reading Sampled Devices

6.2.5.1 READ_LOCATOR_2 (LOCATOR_NUM,X,Y)
READ_LOCATOR_3 (LOCATOR_NUM,X,Y,Z)
(NI)JRDLOC

6.2.5.2 READ_VALUATOR (VALUATOR_NUM,VALUE)
(NI)JRVAXX

6.2.6 Event Handling

6.2.6.1 AWAIT_EVENT (TIME,EVENT_CLASS,EVENT_NUM)
(NI)JREVXX

6.2.6.2 FLUSH_DEVICE_EVENTS (EVENT_CLASS,EVENT_NUM)
(NI)JSEVDV

6.2.6.3 FLUSH_GROUP_EVENTS (EVENT_CLASS,EVENT_NUM_ARRAY,N)
(NI)JSEVCS

6.2.6.4 FLUSH_ALL_EVENTS ()
(NI)JSEVZZ

6.2.7 Associating Devices

6.2.7.1 ASSOCIATE (EVENT_CLASS,EVENT_NUM,SAMPLED_CLASS, SAMPLED_NUM)
(NI)JCEVXX

6.2.7.2 DISASSOCIATE (EVENT_CLASS,EVENT_NUM, SAMPLED_CLASS,SAMPLED_NUM)
(NI)JDEVXX

6.2.7.3 DISASSOCIATE_DEVICE (DEVICE_CLASS,DEVICE_NUM)
(NI)JDDVXX

6.2.7.4 DISASSOCIATE_GROUP(DEVICE_CLASS,DEVICE_NUM_ARRAY,N)
(NI)JDCSGL

6.2.7.5 DISASSOCIATE_ALL ()
(NI)JDDVWW

6.2.8 Accessing Event Report Data

6.2.8.1 GET_PICK_DATA (SEGMENT_NAME,PICK_ID)
(NI)JIIDDA

6.2.8.2 GET_KEYBOARD_DATA (INPUT_STRING,NUM_INPUT)
(NI)JIKBEV

6.2.8.4 GET_LOCATOR_DATA_2 (LOCATOR_NUM,X,Y)
GET_LOCATOR_DATA_3 (LOCATOR_NUM,X,Y,Z)
(NI)JILCDA

6.2.8.5 GET_VALUATOR_DATA (VALUATOR_NUM,VALUE)
(NI)JIVADA

6.2.10 Device Echoing

6.2.10.1 SET_ECHO (DEVICE_CLASS,DEVICE_NUM,ECHO_TYPE)
(NI)JSEOXX

6.2.10.2 SET_ECHO_GROUP (DEVICE_CLASS,DEVICE_NUM_ARRAY,N, ECHO_TYPE)
(NI)JSEOCS

6.2.10.4 SET_ECHO_SURFACE (DEVICE_CLASS,DEVICE_NUM, SURFACE_NAME)
(NI)JSEOPE

6.2.10.5 SET_ECHO_POSITION (DEVICE_CLASS,DEVICE_NUM,ECHO_X, ECHO_Y)
(NI)JSEOCP

6.2.11 Setting Input Device Characteristics

```
6.2.11.2  SET_KEYBOARD (KEYBOARD_NUM,BUFFER_SIZE,
                        INITIAL_STRING,CURSOR_START)
      (NI)JSKBXX

6.2.11.3  SET_BUTTON (BUTTON_NUM,PROMPT_SWITCH)
      (NI)JSBTXX

6.2.11.4  SET_ALL_BUTTONS (PROMPT_SWITCH)
      (NI)JSBTGL

6.2.11.6  SET_LOCATOR_2 (LOCATOR_NUM,LOC_X,LOC_Y)
          SET_LOCATOR_3 (LOCATOR_NUM,LOC_X,LOC_Y,LOC_Z)
      (NI)JSLCXX

6.2.11.8  SET_VALUATOR (VALUATOR_NUM,INITIAL_VALUE,
                        LOW_VALUE,HIGH_VALUE)
      (NI)JSVAXX
```

7.0 Control

7.2.1 Initialization and Termination

```
7.2.1.1  INITIALIZE_CORE (OUTLEVEL,INLEVEL,DIMENSION)
         JNGCXX (LEVEL)

7.2.1.2  TERMINATE_CORE ()
         JEGCXX (LVSN)
```

7.2.2 View Surface Initialization and Selection

```
7.2.2.1  INITIALIZE_VIEW_SURFACE (SURFACE_NAME)
         JNVSXX (LVSN)

7.2.2.2  TERMINATE_VIEW_SURFACE (SURFACE_NAME)
         JEVSXX (LVSN)

7.2.2.3  INQUIRE_OUTPUT_CAPABILITIES (SURFACE_NAME,LEVELS,
                 PHYSICAL,SIZES,PRIM_ATTR,SEG_ATTR,BATCHING)
     (NI)JIOTGL

7.2.2.4  SELECT_VIEW_SURFACE (SURFACE_NAME)
         JCVSXX (LVSN)

7.2.2.5  DESELECT_VIEW_SURFACE (SURFACE_NAME)
         JDVSXX (LVSN)

7.2.2.6  INQUIRE_SELECTED_SURFACES (ARRAY_SIZE,
                   VIEW_SURFACE_NAMES,NUMBER_OF_SURFACES)
         JIVSLS (IARRAY,LARRAY,NLVS)
```

7.2.4 Frame Control

7.2.4.1 NEW_FRAME ()
JXGLFR (DUMMY)

7.2.5 Error Handling

7.2.5.1 REPORT_MOST_RECENT_ERROR (ERROR_REPORT)
JIERXX (LOGERNO,IERCL,IERMES,LEN)

II J-Functions With No Corresponding Core Functions

This section contains a listing of the J-level GRAFCORE functions for which there are no corresponding Core functions. The function names are grouped using the same numbering system as in Section I. The Core Standard section numbers are extended where necessary so that the J-functions can be included within the most appropriate section. Within each group, the functions are listed alphabetically.

2.0 Output Primitives

2.2.5 Raster Extensions - Polygon

JSPGLS (LIS,N) - set polygon edge list
JSPMA3 (XARRAY,YARRAY,ZARRAY,N) - set polygon mesh absolute 3

2.2.7 Grid Drawing Routines

JPGR2D (MAPT,LMAPT) - plot reference grid
JSGR2D (MAPPTYP) - set grid map type

2.2.8 Axis Drawing Routines

JPLSLN (AX,AY,ILEN,FDX,FDY) - plot a list of lines
JPLSTX (AX,AY,MTEXT,ILEN,FDX,FDY) - plot a list of text

2.2.9 Line Bufering Routines

JEBFLN (ICALLER,IBUFNO) - end a line buffer
JNBFLN (ICALLER,IBUFNO,IWS,ILOC,ILEN) - initialize a line buffer
JPBFLN (IBUFNO) - plot a line buffer
JSBFLN (IBUFNO,AX,AY,ILEN) - set lines into a line buffer

3.0 <u>Picture Segmentation and Naming</u>

3.2.1 <u>Retained Segments</u>

JISGCN (LVSN,ISNL,LSN) - inquire segments connected (to view surface)
JISGNO (NUMSEG) - inquire number of segments

4.0 <u>Attributes</u>

4.2.1 <u>Static Attributes</u>

4.2.1.1 Setting Primitive Static Attribute Values

JSTXAN (ORIENT) - set character angle
JSTXBY (IBYTE) - set character byte

4.2.1.3 Inquiring Primitive Static Attribute Values

JITXBY (IBYTE) - inquire character byte

5.0 <u>Viewing Operations and Coordinate Transformations</u>

5.2.1 <u>2-D Viewing Operation</u>

*JXVT2D (VTRANS) - compute viewing transformation for direct mode

5.2.1.5 Inquiry for Individual Viewing Operation Parameters

JIVT2D (VPARS) - inquire viewing transformation

5.2.3 <u>Viewing Control</u>

JSHDXX (IHID) - set hidden lines

7.0 <u>Control</u>

7.2.1 <u>Initialization and Termination</u>

JIGLMD (MODE) - inquire library mode (segment or direct)
JNFT00 (DUMMY) - initialize font 0
JNFT01 (DUMMY) - initialize font 1
*JNFT02-25 (DUMMY) - initialize fonts 2 thru 25
JNFTAH (DUMMY) - initialize Hershey text
JSDIMD (MODE) - set direct mode
JSSGMD (MODE) - set segment mode
JSSGMX (MAXSEG) - set maximum segments

7.2.2 View Surface Initialization and Selection

```
 JCDVAP (IPDPAR,NPAR) - connect a specific device to
                        program
*JCFRAP (IPDPAR,NPAR) - connect a specific FR80 to
                        program
*JCNPAP (IPDPAR,NPAR) - connect a specific NIP to
                        program
*JCPFXX (INARRAY,IARRAY) - connect a picture file
*JCTVAP (IPDPAR,NPAR) - connect a specific TMDS to
                        program
 JCVSDV (NAMPD,LVSN,IOPT) - connect view surface to
                            a device
*JNPFXX (INARRAY,IARRAY) - initialize a picture file
 JSSGVS (LVSN,NAMSEG,IOPT) - set segment (connect/
                         disconnect) to view surface
```

7.2.4 Frame Control

```
JPVSXX (LVSN) - plot (all segments on) view surface
JXVSFR (LVSN) - execute fram action for vs
```

7.2.5 Error Handling

```
JSADBD (IPMIN,IPMAX) - set program address bounds
JSDBMD (JDEBUG) - set debug mode
JSERMG (IEROUT) - set disposition of error messages
JSERZZ (JERFLAG) - set error flag to zero
JSPRXX (JPAR) - set parameter checking
JSRNXX (JRUN) - set run time error checking
JSTCMD (IWRTTR) - set run time trace printing
```

7.2.6 Setting of Defaults

```
JSBFRE (ICLOB) - set buffer reuse flag
JSBFSZ (ICHOMP) - set buffer (chomp) size
JSDVNO (IMAXPD) - set number of devices
JSSGDF (IRESET) - set segment defaults
JSVSNO (IMAXLVSL) - set number of view surfaces
```

8.0 Special Interfaces

8.2.3 Reading and Writing Special Application Data

```
JRAPDA (NAMSEG,DATA,LDATA,MDATA) - read application
                                   data
JWAPDA (NAMSEG,DATA,LDATA) - write application data
```

8.2.4 Device Drivers

```
 JPSGDV (NAMSG,NAMPD) - plot segment to device
*JXFRDR (NAMPD,IOPT,LSN) - execute FR80 driver
*JXNPDR (NAMPD,IOPT,LSN) - execute NIP driver
 JXOTDR (NAMPD,IOPT,LSN) - execute output driver
*JXTVDR (NAMPD,IOPT,LSN) - execute TMDS driver
```

III Core Functions With No Corresponding J-Functions

This section contains a listing of the Core System functions for which there are no designated J-level GRAFCORE functions. The functions are grouped and numbered just as they appear in the Core Standard proposal [Ref 20].

4.0 Attributes

4.2.1 Static Attributes

```
4.2.1.4  INQUIRE_PRIMITIVE_ATTRIBUTES_2
                              (PRIMITIVE_ATTRIBUTE_ARRAY_2)
         INQUIRE_PRIMITIVE_ATTRIBUTES_3
                              (PRIMITIVE_ATTRIBUTE_ARRAY_3)
```

4.2.2 Retained Segment Dynamic Attributes

4.2.2.1 Setting Retained Segment Dynamic Attribute Values

```
SET_VISIBILITY (VISIBILITY)
SET_HIGHLIGHTING (HIGHLIGHTING)
SET_DETECTABILITY (DETECTABILITY)
SET_IMAGE_TRANSLATE_2 (TX,TY)
SET_IMAGE_TRANSFORMATION_2 (SX,SY,A,TX,TY)
SET_IMAGE_TRANSFORMATION_3 (SX,SY,SZ,AX,AY,AZ,TX,
                                             TY,TZ)
```

4.2.2.2 Inquiring Retained Segment Dynamic Attribute Values

```
INQUIRE_VISIBILITY (VISIBILITY)
INQUIRE_HIGHLIGHTING (HIGHLIGHTING)
INQUIRE_DETECTABILITY (DETECTABILITY)
INQUIRE_IMAGE_TRANSLATE_2 (TX,TY)
INQUIRE_IMAGE_TRANSFORMATION_2 (SX,SY,A,TX,TY)
INQUIRE_IMAGE_TRANSFORMATION_3 (SX,SY,SZ,AX,AY,
                                      AZ,TX,TY,TZ)
```

6.0 Input Primitives

6.2.4 Initializing and Enabling Input Devices

6.2.4.1 INITIALIZE_DEVICE (DEVICE_CLASS,DEVICE_NUM)

6.2.4.2 INITIALIZE_GROUP (DEVICE_CLASS,DEVICE_NUM_ARRAY,N)

6.2.4.8 TERMINATE_DEVICE (DEVICE_CLASS,DEVICE_NUM)

6.2.4.9 TERMINATE_GROUP (DEVICE_CLASS,DEVICE_NUM_ARRAY,N)

6.2.8 Accessing Event Report Data

6.2.8.3 GET_STROKE_DATA_2 (ARRAY_SIZE,X_ARRAY,Y_ARRAY, NUM_POSITIONS)
GET_STROKE_DATA_3 (ARRAY_SIZE,X_ARRAY,Y_ARRAY, Z_ARRAY,NUM_POSITIONS)

6.2.9 Synchronous Input Functions

6.2.9.1 AWAIT_ANY_BUTTON (TIME_BUTTON_NUM)

6.2.9.2 AWAIT_PICK (TIME,PICK_NUM,SEGMENT_NAME,PICK_ID)

6.2.9.3 AWAIT_KEYBOARD (TIME,KEYBOARD_NUM,INPUT_STRING, LENGTH)

6.2.9.4 AWAIT_STROKE_2 (TIME,STROKE_NUM,ARRAY_SIZE,X_ARRAY, Y_ARRAY,NUM_POSITIONS)
AWAIT_STROKE_3 (TIME,STROKE_NUM,ARRAY_SIZE,X_ARRAY, Y_ARRAY,Z_ARRAY,NUM_POSITIONS)

6.2.9.5 AWAIT_ANY_BUTTON_GET_LOCATOR_2 (TIME,LOCATOR_NUM, BUTTON_NUM,X,Y)
AWAIT_ANY_BUTTON_GET_LOCATOR_3 (TIME,LOCATOR_NUM, BUTTON_NUM,X,Y,Z)

6.2.9.6 AWAIT_ANY_BUTTON_GET_VALUATOR (TIME,VALUATOR_NUM, BUTTON_NUM,VALUE)

6.2.10 Device Echoing

6.2.10.3 SET_ECHO_SEGMENT (DEVICE_CLASS,DEVICE_NUM, SURFACE_NAME)

6.2.11 Setting Input Device Characteristics

6.2.11.1 SET_PICK (PICK_NUM,APERTURE)

6.2.11.5 SET_STROKE (STROKE_NUM,BUFFER_SIZE,DISTANCE,TIME)

6.2.11.7 SET_LOCPORT_2 (LOCATOR_NUM,XMIN,XMAX,YMIN,YMAX)
SET_LOCPORT_3 (LOCATOR_NUM,XMIN,XMAX,YMIN,YMAX,
ZMIN,ZMAX)

6.2.12 Inquiry

6.2.12.1 INQUIRE_INPUT_CAPABILITIES (LEVEL,DEVICE_COUNTS,
TIMING)

6.2.12.2 INQUIRE_INPUT_DEVICE_CHARACTERISTICS
(DEVICE_CLASS,DEVICE_NUM,IMPLEMENTATION,ECHO,
VIEW_SURFACE,ASSOCIATION_SIZE,ASSOCIATION_CLASS,
ASSOCIATION_NUM,ASSOCIATION_COUNT,
DUPLICATION_SIZE,DUPLICATION_CLASS,
DUPLICATION_NUM,DUPLICATION_NUM,
DUPLICATION_COUNT,PRECISION,DELAY)

6.2.12.3 Inquire Dimension

INQUIRE_STROKE_DIMENSION (STROKE_NUM,DIMENSION)
INQUIRE_LOCATOR_DIMENSION(LOCATOR_NUM,DIMENSION)

6.2.12.4 INQUIRE_DEVICE_STATUS (DEVICE_CLASS,DEVICE_NUM,
INITIALIZED,ENABLED)

6.2.12.5 INQUIRE_DEVICE_ASSOCIATIONS(EVENT_CLASS,EVENT_NUM,
ARRAY_SIZE,SAMPLED_CLASS_ARRAY,SAMPLED_NUM_ARRAY,
NUMBER_OF_ASSOCIATIONS)

6.2.12.6 Inquire Input Status Parameters

INQUIRE_ECHO (DEVICE_CLASS,DEVICE_NUM,ECHO_TYPE)
INQUIRE_ECHO_SURFACE (DEVICE_CLASS,DEVICE_NUM,
SURFACE_NAME)
INQUIRE_ECHO_POSITION (DEVICE_CLASS,DEVICE_NUM,
ECHO_X,ECHO_Y)
INQUIRE_PICK (PICK_NUM,APERTURE)
INQUIRE_KEYBOARD (KEYBOARD_NUM,BUFFER_SIZE,
INITIAL_STRING,CURSOR_START)
INQUIRE_BUTTON (BUTTON_NUM,PROMPT_SWITCH)
INQUIRE_STROKE (STROKE_NUM,BUFFER_SIZE,DISTANCE,
TIME)
INQUIRE_LOCATOR_2 (LOCATOR_NUM,LOC_X,LOC_Y)
INQUIRE_LOCATOR_3(LOCATOR_NUM,LOC_X,LOC_Y,LOC_Z)

INQUIRE_LOCPORT_2 (LOCATOR_NUM,XMIN,XMAX,YMIN,
YMAX)

INQUIRE_LOCPORT_3 (LOCATOR_NUM,XMIN,XMAX,YMIN,
YMAX,ZMIN,ZMAX)

INQUIRE_VALUATOR (VALUATOR_NUM,INITIAL_VALUE,
LOW_VALUE,HIGH_VALUE)

6.2.12.7 INQUIRE_ECHO_SEGMENTS (DEVICE_CLASS,DEVICE_NUM,
ARRAY_SIZE,ECHO_SEGMENT_ARRAY,
NUMBER_ECHO_SEGMENTS)

7.0 Control

7.2.3 Picture Change Control

7.2.3.2 SET_IMMEDIATE_VISIBILITY (IMMEDIACY)

7.2.3.3 MAKE_PICTURE_CURRENT ()

7.2.3.5 BEGIN_BATCH_OF_UPDATES ()

7.2.3.6 END_BATCH_OF_UPDATES ()

7.2.3.7 INQUIRE_CONTROL_STATUS (IMMEDIACY,BATCHING)

7.2.3.8 SET_VISIBILITIES (SEGMENT_NAME_ARRAY,
VISIBILITY_ARRAY,N)

7.2.5 Error Handling

7.2.5.2 LOG_ERROR (ERROR_REPORT)

8.0 Special Interfaces

8.2.1 General Escape Functions

8.2.1.1 ESCAPE (FUNCTION_NAME,PARAMETER_COUNT,
PARAMETER_LIST)

8.2.1.2 INQUIRE_ESCAPE (FUNCTION_NAME,SUPPORT_INDICATOR)

8.2.2 Specific Escape Functions

8.2.2.1 ESCAPE (PROMPT,4,PARAMETER_LIST)

Note: None of the raster extensions are implemented by GRAFCORE except for POLYGON_ABS_3.

Appendix B

GRAFCORE K-level Functions

This appendix contains a listing of the GRAFCORE K-level function names and the operations that they perform. The K-functions are internal routines that support the J-level functions of GRAFCORE. No K-level routine calls a J-level routine. The function names are grouped according to purpose. Within each group, the functions are listed alphabetically. Functions which are not included in our version of the GRAFCORE package are marked with an asterisk ('*').

Contents

Contents

1. Text Routines

KINOTX - Inquire numbers to text options
KSNOTX - Set numbers to text options
KTA6A8 - Transform 6-bit to 8-bit
KTA8A6 - Transform 8-bit to 6-bit
KTNOEE - Transform numbers to text (e format)
KTNOFF - Transform numbers to text (f format)
KTNOGG - Transform numbers to text (g format)
KTNOII - Transform numbers to text (i format)
KTNOTX - Transform numbers to text
KTSCTX - Transform integer powers to text
KTTXOC - Transform bcd to octal

2. Hershey Character Routines

KIAHPL - Inquire Hershey lines
KPAHPL - Plot Hershey character (as lines)
KSAHDA - Set Hershey data
KSAHFT - Set Hershey font
KTTXAH - Transform 8-bit to Hershey
KXDRAH - Execute code for Hershey text

3. Grid Routines

KINCLG - Inquire nice log values
KPGR2D - Plot division and tick labels
KPGRLN - Plot grid tick marks and/or lines

4. Axis Routines

KILSLN - Inquire list of line options
KILSTX - Inquire list of text options
KSLSLN - Set list of line options
KSLSTX - Set list of text options
KTNOSC - Transform numbers to be scaled
KXNCLG - Compute nice log numbers
KXNCNO - Compute nice numbers

5. Line Buffering Routines

KIBFLN - Inquire line buffering options
KSBFLN - Set line buffering options

6. Segment Routines

KISGST - Inquire segment state
KRSGBF - Read from segment to buffer
KRSGXX - Read segment
KWBFSG - Write buffer to segment

7. Directory Routines

KIDEXX - Inquire directory entry
KISGDE - Inquire segment directory
KNDEXX - Initialize directory entry
KNSGDE - Initialize segment directory
KSSGDE - Set segment diretory entry

8. Attribute Routines

KIATRN - Inquire run time attribute
KSATRN - Set run time attribute
KSFSYS - Define font parameters
KSSGAT - Set attribute in segment
KSTXJF - Set text justification offsets
KTTXHI - Get text location
KXLISZ - Compute line size
KXLNFT - Compute line style

9. Transformation and Matrix Routines

KIDTDV - Inquire device transformation and device type
KIVT2D - Inquire viewing transformation matrix
KSITXX - Set image transformation
KSMXRT - Set rotation matrix
KSMXSC - Set scaling matrix
KSMXTL - Set translation matrix
KSMXZZ - Set matrix to zero
KSVT2D - Set 2-D viewing transformation matrix
KSVT3D - Set 3-D viewing transformation matrix
KTSC2D - Transform data (scale to ndc)
KVMXID - Verify for identity matrix
KXAN3D - Compute sin/cos
KXIT2D - Execute image transformation
KXMXMX - Execute matrix * matrix
KXVCMX - Execute vector * matrix
KXVT2D - Execute viewing transformation

10. Clipping Routines

KXCL2D - Execute clip points to window
KXCL3D - Execute clip edges to view frustum
KXCLPE - Execute clip an edge to an arbitrary plane

11. Logical View Surface Routines

KSVSMD - Set view surface mode (active/inactive)

12. System Routines

KIAPLV - Inquire application program level
KNMGBF - Initialize message buffer
KSMGBF - Set message buffer to zero
KSMGJF - Set message justify option
KXMGIO - Execute send message and get response
KXTXPK - Execute squeeze blanks
KXTXQ1 - Execute inquire input(support KXTXPK)
KXTXQ2 - Execute store input (support KXTXPK)
KXTXQ3 - Execute store output (support KXTXPK)

13. Initializing Common

KBBFLN - Block data for line buffering
KBBFLS - Block data for line buffers
KBDAT1 - Block data for GRAFCORE flags
KBDAT2 - Block data for default attributes
KBDAT3 - Block data for picture file op codes
KBDAT4 - Block data for workspace connectors and grafpar parameters
KBDFAH - Block data default Hershey text
KBFT01 - Block data for font 1
KBLSLN - Block data for list of lines
KBLSTX - Block data for list of text
KBNOTX - Block data for number conversion
KNGCXX - Initialize attribute common
KNPFGC - Initialize picture file common
KSPRGC - Set parameter common

14. Buffer Routines

KCBFXX - Connect buffer
KDBFXX - Disconnect buffer
KEBFXX - End buffer
KIBFCN - Inquire free buffer connector
KIBFSP - Inquire buffer space
KIBFST - Inquire buffer status and descriptor
KIBFXX - Inquire buffer
KNBFXX - Initialize buffers and workspace
KSBFDE - Set (update) buffer directory
KSBFZZ - Set buffer to zero
KVBFAD - Verify buffer address

15. Display List Intermediate Routines (DLI)

KIPFDA - Inquire next picture file data
KIPFOP - Inquire picture file operation
KXPFPK - Execute squeeze out nonretained segments

16. DLI File Functions

*KRPFBF - Read picture file to buffer
*KWBFPF - Write buffer to picture file
KWSGDE - Write segment directory

17. Disk File Functions

KCFNXX - Connect (open) a file
KDFNXX - Disconnect (close) a file
KEUSXX - End (release) the unit specifier
*KIFNXX - Initialize (create) a file
KIUSXX - Inquire free unit specifier
KMFNFN - Move (copy) file to file
KNFNNM - Initialize (compose) a file name
KRFNBF - Read from file to buffer
KWBFFN - Write from buffer to file

18. External Linkage

KXJMP1 - Execute external jump
KXJMP2 - Execute external jump
KXJMP3 - Execute external jump
KXJMP4 - Execute external jump
KXJMP5 - Execute external jump

19. Error Routines

KMERLS - Move error to list
KSER3D - Set error for 3D
KSERMG - Set (format) an eror message
KSTCLS - Set trace list
KVGLXX - Verify "graflib" for clobbered common
KVPRMM - Verify parameter against min/max
KVPRNO - Verify number of parameters
KXERXX - Execute (fatal) error code

20. TMDS Device Driver Routines

*KCTVBF - Connect users TMDS buffer
*KCTVDV - Connect TMDS device
*KDTVDV - Disconnect TMDS device
*KETVDV - End TMDS device
*KITVER - Inquire TMDS error
*KITVQ7 - Inquire TMDS status
*KITVST - Inquire TMDS state information
*KMTVDA - Move TMDS data to buffer
*KMTVFO - Move TMDS text to buffer
*KMTVQ8 - Move TMDS commands to buffer
*KNTVDV - Initialize TMDS device
*KNTVST - Initialize TMDS state list
*KPTVFO - Plot TMDS hardware text
*KPTVLN - Plot lines, points for TMDS
*KRTVDV - Read message from TMDS
*KSTVER - Set TMDS error
*KSTVFO - Set TMDS hardware text parameters
*KSTVFR - Set frame advance TMDS
*KSTVQ1 - Set TMDS control register
*KSTVQ2 - Set TMDS raster location
*KSTVQ3 - Set TMDS additive mode
*KSTVQ4 - Set TMDS polarity
*KSTVQ5 - Set TMDS prompt option
*KSTVQ6 - Set TMDS replacement mode
*KSTVST - Set TMDS state information
*KSTVTX - Set TMDS text properties
*KSTVUS - Set TMDS unit specifier
*KSTVZZ - Set TMDS buffer to zero
*KVTVBF - Verify TMDS buffer
*KVTVLV - Verify and confirm TMDS level
*KWTVBF - Write TMDS buffer
*KWTVDA - Write TMDS data
*KWTVDV - Write message to TMDS
*KWTVXX - Write TMDS picture
*KXTVFR - Execute TMDS frame advance
*KXTVQU - Execute queue service for TMDS
*KXTVTX - Compute TMDS text

21. NIP Device Driver Routines

*KDNPFN - Disconnect (close) the NIP file
*KENPDV - End NIP device
*KINPSP - Inquire NIP file space
*KINPST - Inquire NIP state information
*KMNPDA - Move NIP data to buffer
*KNNPDV - Initialize NIP device
*KNNPQ1 - Initialize next NIP file
*KPNPFO - Plot NIP hardware characters
*KPNPFT - Plot NIP test pattern
*KPNPID - Plot NIP identification page
*KPNPLB - Plot label on the NIP frame
*KPNPLN - Plot lines, points for NIP
*KSNPFO - Initialize NIP hardware characters
*KSNPST - Set NIP state information
*KWNPFN - Write NIP file
*KWNPID - Write NIP file information block
*KXNPFR - Execute NIP frame advance
*KXNPPL - Execute functions for NIP vectors

22. FR80 Device Driver Routines

*KEFRDV - End FR80 device
*KIFRST - Inquire FR80 state information
*KMFRDA - Move FR80 data to buffer
*KNFRDV - Initialize FR80 device
*KPFRFO - Plot FR80 hardware text
*KPFRID - Plot FR80 identification pages
*KPFRLB - Plot FR80 chaacters
*KPFRLN - Plot lines, points for FR80
*KPFRTX - Plot FR80 hardware characters
*KSFRCR - Set FR80 color
*KSFRFO - Set FR80 hardware text parameters
*KSFRIN - Set FR80 intensity
*KSFRQ1 - Set color hit count
*KSFRQ3 - Set FR80 repeat
*KSFRST - Set FR80 state information
*KWFRBF - Write FR80 buffer to file
*KXFRFR - Execute FR80 frame advance
*KXFRPK - Execute pack FR80 commands

23. Device Dependent Functions

KNTXDR - Initialize text
KSDVAT - Set (output) device attribute
KSDVST - Set device state list
KXDRFO - Execute code for hardware text
KXDRF1 - Execute code for font1 text
KXDRPL - Execute line drawer

24. Statistic Routines

KIGCNO - Inquire GRAFCORE statistics
KSGCNO - Set GRAFCORE statistics to zero

25. Miscellaneous

KXBIXX - Compute run of bits

Vita

Philip Boman Tarbell, III was born on 28 December 1947 in New Haven, Connecticut. He graduated from Springbrook High School, Silver Spring, Maryland, in 1965 and then attended Montgomery Junior College, Takoma Park, Maryland, from which he received an Associate degree in Electronic Technology in 1968. In January 1970, he enlisted in the Air Force and became an ECM simulator technician. He was accepted into the Airman Education and Commissioning Program and assigned to the University of Maryland from August 1974 to August 1976, graduating with a Bachelor of Science degree in Electrical Engineering. He then attended the Air Force Officer Training School at Lackland AFB, Texas from September 1976 to December 1976. Following his commissioning, he was assigned to Headquarters, USAF Security Service, as a MAJCOM Systems Project Engineer. He entered the Air Force Institue of Technology's masters degree program in June 1980. He is a member of Eta Kappa Nu and IEEE.

Permanent Address: 9313 New Hampshire Avenue
Silver Spring, Maryland 20903

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFIT/GE/EE/81D-58	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): CONTINUED DEVELOPMENT AND IMPLEMENTATION OF A STANDARD GRAPHICS PACKAGE FOR THE AFIT VAX 11/780	5. TYPE OF REPORT & PERIOD COVERED: MS Thesis
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Philip B. Tarbell, Captain, USAF	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433	12. REPORT DATE: December 1981
	13. NUMBER OF PAGES: 161
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

15 APR 1982

18. SUPPLEMENTARY NOTES

Approved for public release, IAW AFR 190-17

Frederic C. Lynch, Major, USAF
Director of Public Affairs

Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Graphics
Core Standard
GRAFCORE
GWCORE
VAX 11/780

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See Reverse

DD FORM 1473, 1 JAN 73 — EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20.

Abstract

Graphics packages that have been implemented based on the 1979 ACM/SIGGRAPH Core Standard Proposal were investigated. Two packages were identified that best met AFIT's needs, based on host processor implementation and availability. These two packages were GRAFCORE, from the Lawrence Livermore National Laboratory, and GWCORE, from the George Washington University. GRAFCORE was the package initially selected for installation on the AFIT VAX. GWCORE was also obtained, but not installed. Both packages currently provide output-only capability. Problems with GRAFCORE installation that had to be overcome included characters lost in transmission, incomplete files, and the need to manually convert some of the code from the LRLTRAN language used at Lawrence Livermore into standard FORTRAN. Future efforts in this area should begin with device driver design and implementation for both graphics packages, followed by implementation of graphics-input routines.

FILMED

7-8